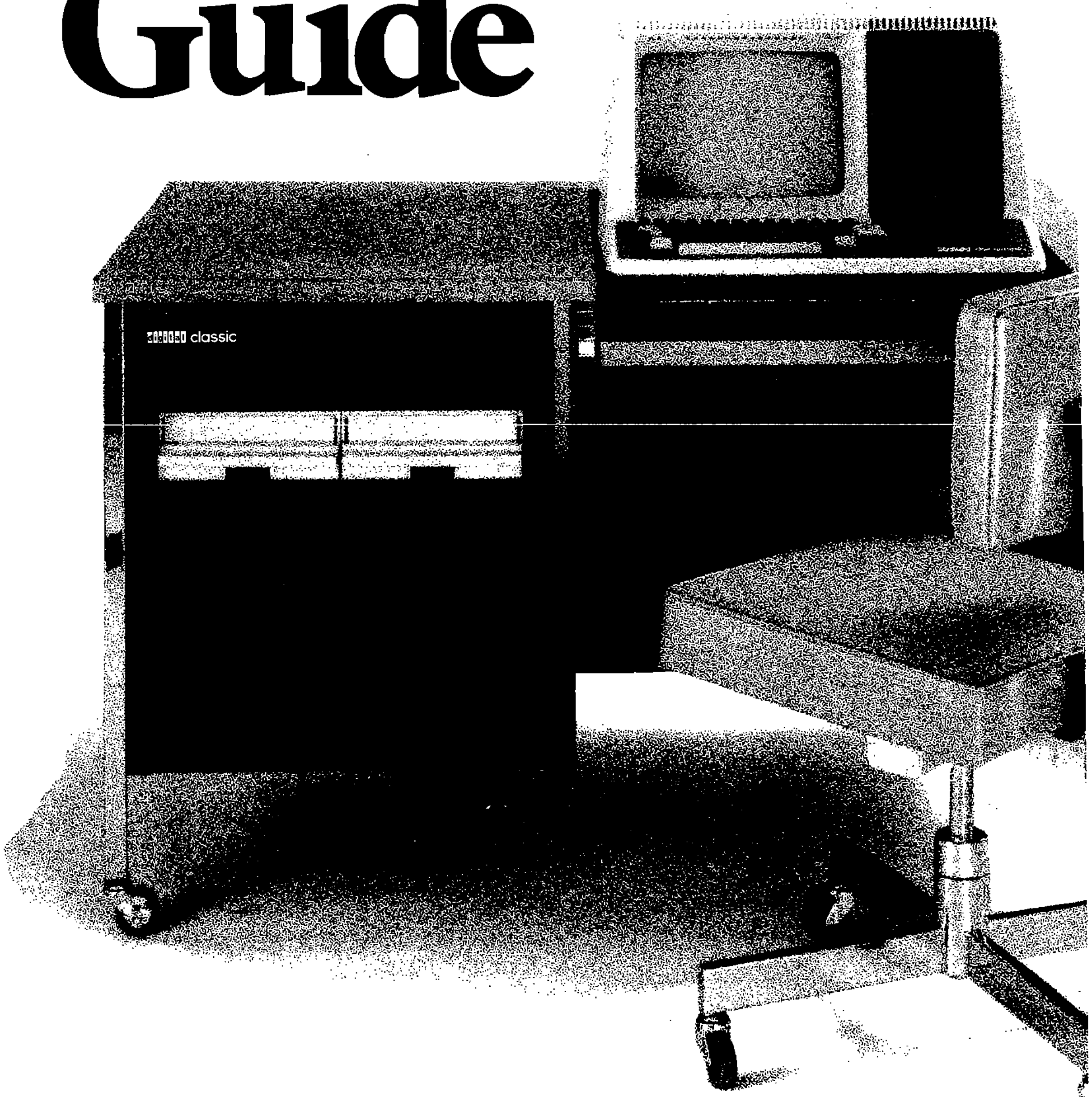
The Classic Primer: A Self-Teaching Guide



digital

THE CLASSIC PRIMER:

A SELF-TEACHING GUIDE

DEC-08-ECPGA-B-D

PREPARED
BY

COURSE DEVELOPMENT GROUP

EDUCATIONAL SERVICES DEPARTMENT

DIGITAL EQUIPMENT CORPORATION •
MAYNARD, MASSACHUSETTS

June, 1976
First Printing, May 1975

PREFACE

This book is part of a three-volume set that documents the CLASSIC system. The three volumes in this set are:

(1) *The CLASSIC Primer: A Self-Teaching Guide*
Order No. DEC-08-ECPGA-B-D

(2) *The CLASSIC User's Reference Guide*
Order No. DEC-08-ECUGA-B-D

(3) *The CLASSIC Installation and Maintenance Guide*
Order No. DEC-08-ECIMA-B-D

The *Primer* is designed to assist the novice user in learning to operate CLASSIC and write programs in the BASIC language. The *User's Reference Guide* consists primarily of alphabetical directories of all the commands recognized by CLASSIC with explanations and examples of each command. The *Installation and Maintenance Guide* provides step by step guidance for installing the CLASSIC system and a detailed procedure for correcting minor problems.

Table of Contents

Chapter		Page
1	RUNNING CLASSIC	1-1
	How To Run a CLASSIC Program	1-1
	Selected Programs on the BASIC Program Demonstration Disk	1-8
2	USING CLASSIC	2-1
	What Is CLASSIC?	2-1
	Using the CLASSIC Software	2-3
	Typing Rules Used in This Guide	2-3
3	BEGINNING BASIC PROGRAMMING	3-1
	Understanding What To Do	3-1
	3-A Calculating	3-1
	3-B Printing Larger Numbers and Words	3-5
	3-C Printing Variable Results	3-9
	3-D Editing Larger Programs	3-15
	3-E Using Disk Files	3-19
	3-F Loops	3-22
	3-G Creating FOR-NEXT Loops	3-28
	3-H Supplying Larger Amounts of Data	3-22
	3-I Organizing Your Programs	3-39
4	ADVANCED BASIC PROGRAMMING	4-1
	4-A Numeric Functions	4-1
	4-B Alphanumeric and Special Functions (Part I)	4-10
	4-C Alphanumeric and Special Functions (Part II)	4-19
	4-D Storing Data in Disk Files	4-28
	4-E Using Monitor Commands	4-36

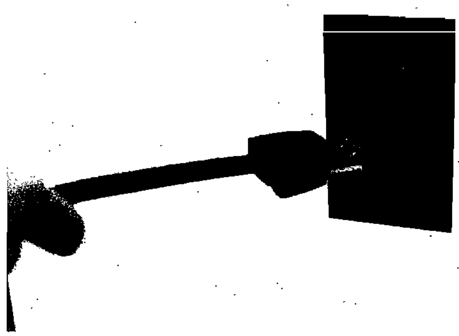
TABLE OF CONTENTS (continued)

Chapter		Page
5	CLASSIC APPLICATIONS	5-1
	Understanding What To Do	5-1
	5-A Examples of CLASSIC Applications	5-2
	5-B Planning Programs for CLASSIC	5-5
	5-C Documenting Your Programs	5-7
	5-D Transporting Your Programs	5-9
	5-E Identifying Further Resources	5-12
ADDENDUM:	USING THE LINE PRINTER	
APPENDIX A	Write-Ups for Applications Programs	A-1
	ACEY02	A-1
	ATTEND and ATTSET	A-3
	CALC	A-5
	EASY02 and EASY03	A-5
	GUESS	A-6
	HMRABI	A-6
	HURKLE	A-7
	HURK02	A-8
	MORGAG	A-10
	QUADEQ, QUAD02, and QUAD03	A-11
	SYNONY and SYNSET	A-12
	WTDAVG	A-13
APPENDIX B	DECUS Program Submission Forms	B-1
APPENDIX C	Answers To Exercises	C-1

Chapter I
Running Classic

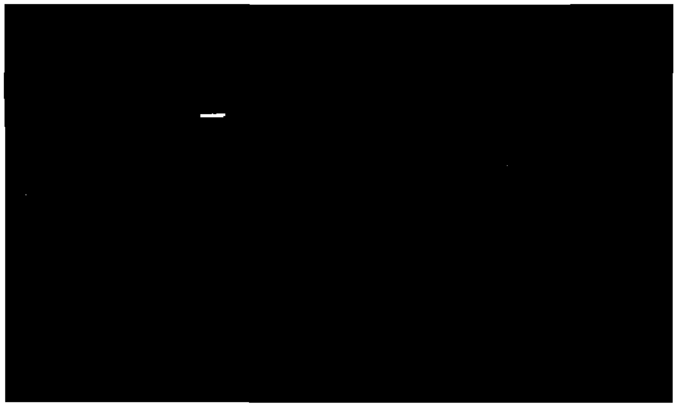
HOW TO RUN A CLASSIC PROGRAM

This chapter will help you learn how to run a computer program on CLASSIC. Start by telling your teacher or instructor that you want to use the computer. Arrange for a time to use CLASSIC and ask him or her to lend you copies of the CLASSIC System disk and the BASIC Program Demonstration disk. Then follow these steps:

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS
① Make sure that the computer is plugged into a 3-holed socket.		
② Push the **top** of the red ON/OFF switch on the front of the machine so that it stays in.		

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

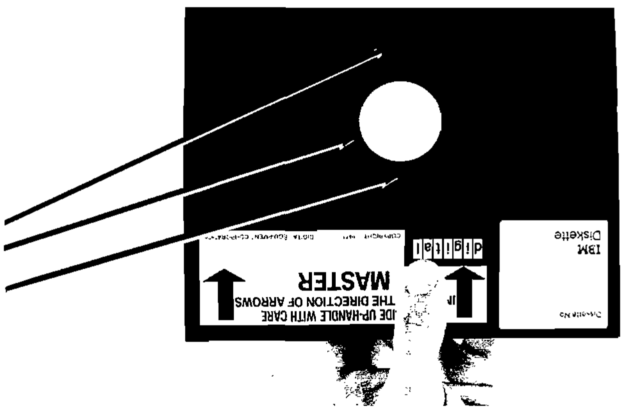
You should now hear some "clicks" from inside the computer. You may even hear the soft whirr of fans. In a minute, you should see a short flashing line on the screen.



If you do not see this line, make sure that the plug is pushed all the way in and press the red ON/OFF switch again. If nothing happens this time, ask your teacher or instructor for help.

③
When you hold a CLASSIC disk, **DO NOT TOUCH THE BROWN PARTS** that appear through the holes in the cover. Always hold a disk only by its cover.

DO **NOT**
TOUCH
HERE

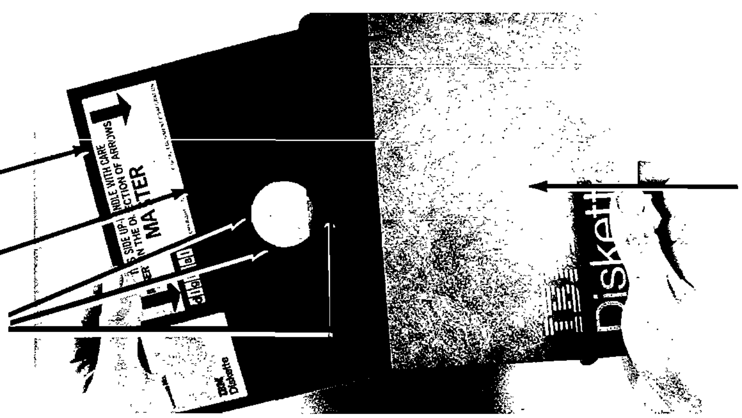


Take the CLASSIC System disk out of its envelope by placing your thumb on the label.

disk label

disk cover

disk surface

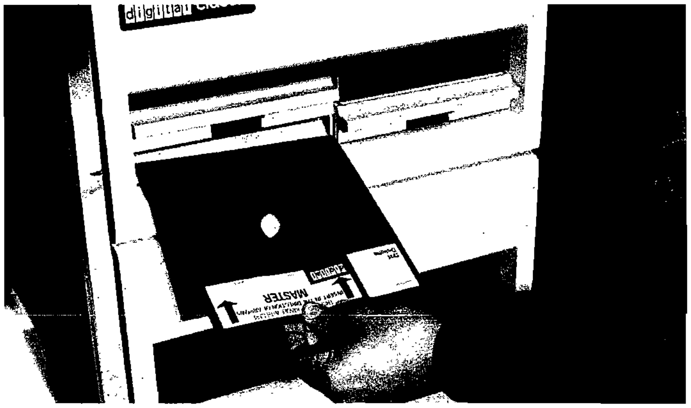


envelope

④
Lift the left-hand door on the front of the CLASSIC by pinching its latch between your thumb and finger.

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

Slide the CLASSIC System disk into the drawer, label side up, but **DO NOT FORCE THE DISK** into the drawer. It should slide in smoothly.



If the disk does not slide in smoothly, make sure that you have lifted the door all the way up and that another disk is not already in the drawer. If you find another disk, slide it out and give it to your teacher or instructor. **DO NOT PLACE THE DISK ON THE DESK WITHOUT ITS ENVELOPE.**

⑤
Close the door over the disk, but **DO NOT FORCE THE DOOR CLOSED.** The door latch will "click" when it is closed properly.

If the door will not close, make sure that the disk is pushed all the way in. If you still cannot close the door, ask your teacher or instructor for help.

⑥
Take the BASIC Program Demonstration disk out of its envelope and slide it into the right-hand drawer. Close the right-hand door over the second disk so that it clicks.

⑦
Push the top of the white START button on the front of the machine and let it go again, allowing it to rock out.

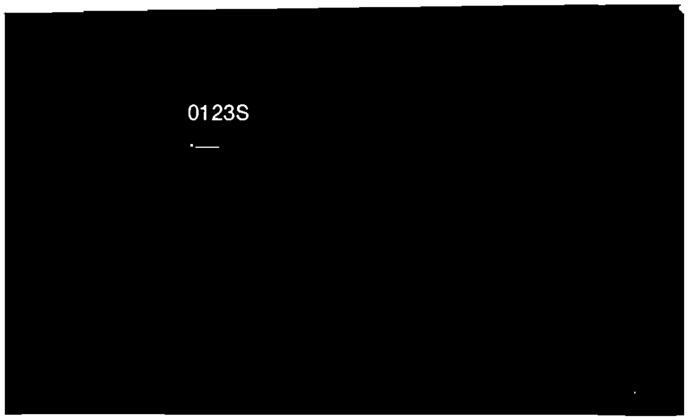
DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

The numbers 0123 should now appear on the screen.

If all of these numbers do not appear, press the white START button again. If they still do not appear, make sure that the left-hand disk is pushed all the way in and that its door is properly closed.

Then push the white START button once again. If you still do not see all the numbers, ask your teacher or instructor for help.

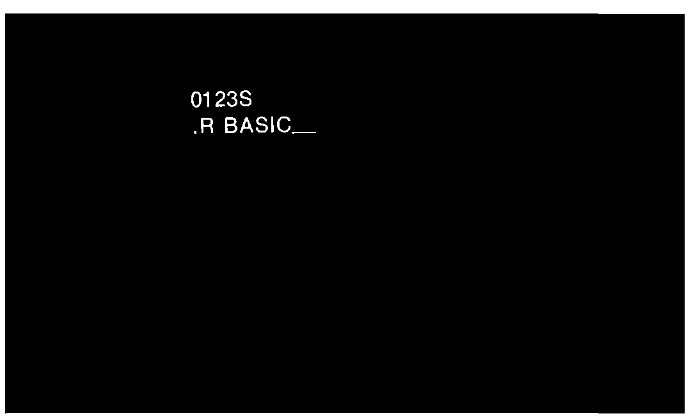
⑧
Press the letter S on the keyboard.
You should see the letter S appear and then CLASSIC should display a dot on the next line.



If the S does not appear on screen, press the S key again. If the dot does not appear, repeat Step 7. Ask your teacher or instructor if you need help.

⑨
Look at the CLASSIC keyboard and find the space bar and the keys that say CTRL, U, and RETURN.

Now type R BASIC after the dot, pressing the space bar once between the R and B as shown in the picture.



If you make a mistake, hold down the CTRL key and press the U key. This will print ∧ U and another dot will appear. Then type R BASIC correctly.

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

Everything you type will appear on the screen. If you make a mistake, hold down the CTRL key and type U to tell CLASSIC to delete the line you have just typed. Then retype the line correctly.

⑩
Push the wide key that says RETURN.

Pushing the RETURN key tells CLASSIC to read the line you have just typed.

CLASSIC should print the message NEW OR OLD—

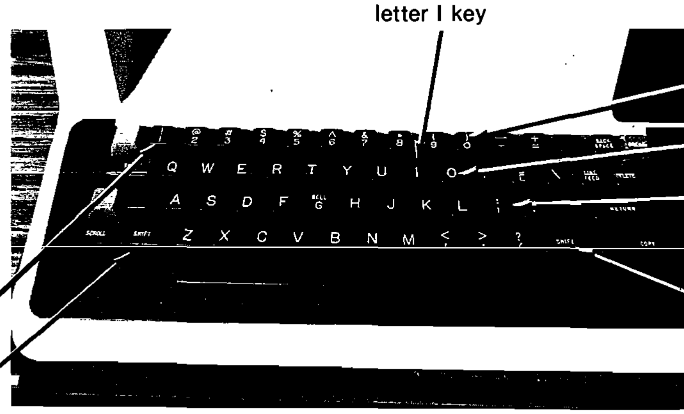
If NEW OR OLD— is not printed, look to see if a new dot has been printed after any other message that you might see. If you see a new dot, type R BASIC again and push RETURN.

If you do not see the NEW OR OLD— message or a new dot, hold down the CTRL key and press C. This should cause CLASSIC to print a new dot. Then type R BASIC and press RETURN.

If NEW OR OLD— still does not appear, ask your teacher or instructor for help.

⑪
Find the keys that say 1 and 0 on the top row of the keyboard. You must always use these keys to type the numbers one and zero. The keys that say I and O on the second row of the keyboard are used to type letters only.

letter I key



number 0 key

letter O key

: key

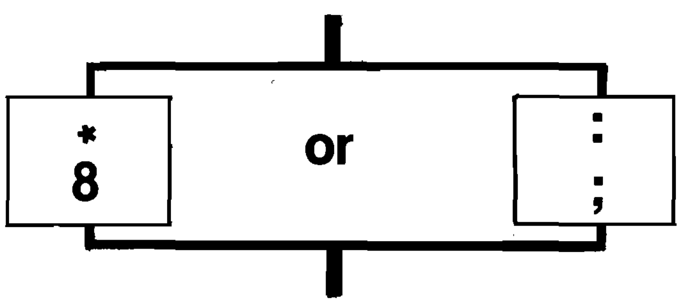
SHIFT key

number 1 key

SHIFT key

On keys that have two characters like the [*/8] and [:/;] keys, you can type the upper character by holding down the SHIFT key while pressing the character key. For example, if you want to type ":", you must hold down the SHIFT key and press the [:/;] key.

These are typed by holding down the SHIFT key.



These are typed normally.

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

⑫
Now type OLD RXA1:GUESS after the NEW OR OLD— message. Press the space bar once between the D and R, and be sure to use the correct keys for 1 and : as shown in the picture.

```
0123S
.R BASIC
NEW OR OLD --OLD RXA1:GUESS

READY

—
```

If you make a mistake, hold down the CTRL key and press U. CLASSIC will respond "DELETED". Then type OLD RXA1:GUESS again.

Press the RETURN key again. CLASSIC should then respond only with the word READY as shown in the figure above.

If the message OLD FILE NAME— is printed, type RXA1:GUESS and press RETURN.

If the message BAD FILE is printed, type OLD RXA1: GUESS again and press RETURN. If BAD FILE is printed again, ask your teacher or instructor for help.

If any other message is printed, make sure that the right-hand door is closed completely and begin again from Step 7. If you have further trouble, ask your teacher or instructor for help.

⑬
Now type:
RUN
and press RETURN. In a minute, CLASSIC should print:
GUESS BA 3.0
and then display messages for you to read and questions for you to answer. The computer will tell you that it is waiting for an answer by printing a question mark (?). Type your answers after the question marks. Do not forget to press the RETURN key after you type to tell CLASSIC to read your answer. If you make a mistake, hold down the CTRL key and press the U key.

```
0123S
.R BASIC
NEW OR OLD__OLD RXA1: GUESS

READY

RUN__
```

If a question mark does not appear after the messages, ask your teacher or instructor for help.

⑭
When you have played GUESS as much as you like, hold down the CTRL key and type C. This will cause the READY message to be displayed again.

```
^C
READY
—
```

Typing C while holding down the CTRL key tells CLASSIC to stop whatever it is doing and let you type new lines.

DO THIS	LIKE THIS	BUT IF SOMETHING GOES WRONG, READ THIS

⑮

Whenever you see the READY message, you can ask CLASSIC to run another program. For example, type:

OLD RXA1:SYNONY

and press RETURN. CLASSIC should respond:

READY

without any other message.

```
READY
OLD RXA1: SYNONY
READY
_
```

If any other message is displayed, type OLD RXA1: SYNONY and press RETURN again. If you have further problems, ask your teacher or instructor for help.

⑯

Now type RUN and press RETURN. In a minute, CLASSIC should print:

SYNONY BA 3.0

and then give you further instructions. SYNONY is a ten question drill on synonyms that records the scores achieved by all the students who use it. This program will end by itself, so you do not have to type CTRL/C.

```
READY
OLD RXA1: SYNONY
READY
RUN
_
```

If further instructions are not displayed, ask your teacher or instructor for help.

⑰

When the READY message reappears, you can ask CLASSIC to run another program if you like. The names of some of the other programs that you can run using the BASIC Program Demonstration disk are:

ACEY02	*HURK02*
CALC	*MORGAG*
EASY03	*QUAD03*
HMRABI	*WTDAVG*
HURKLE	

Each of these programs is explained at the end of this chapter. To use any of these, type OLD RXA1: and the program name as you did for SYNONY in Step 15. For example, you might type:

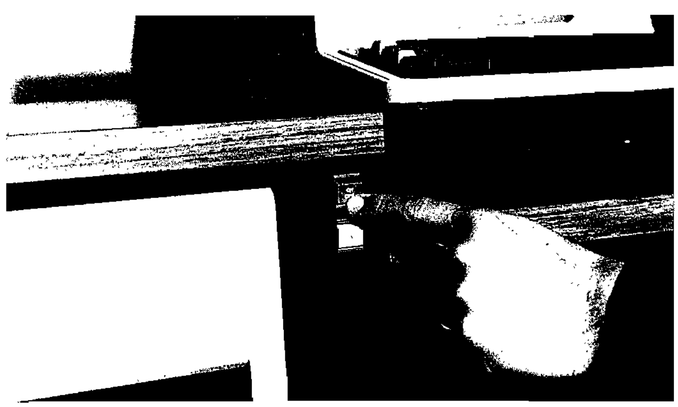
OLD RXA1:ACEY02

and press RETURN. Then type RUN as you did in Step 13.

⑱

When you have finished working with CLASSIC, type CTRL/C as many times as necessary until the dot reappears on the screen at the beginning of a line. Then open the doors over the disks and gently slide the disks out from the drawers. Place the disks back in their envelopes so that the labels can be seen. Hold the disks as you did in Step 3.

When both disk drawers are empty, close their doors and push the **bottom** of the red ON/OFF switch. The display will disappear. Return the disks to your teacher or instructor.



SELECTED PROGRAMS ON THE BASIC PROGRAM DEMONSTRATION DISK

Below are explanations of some of the programs that you can run using the BASIC Program Demonstration disk. Each program is followed by part of a sample run. The lines that you type have been circled. For additional information on these programs, see Appendix A. (Your teacher or instructor may have another disk with additional programs that you can run.)

ACEY02 plays the card game Acey-Deucey. The computer deals two cards and you bet on whether a third card will fall between them. You begin with $100; aces are high and deuces are low.

```
OLD RXA1:ACEY02

READY
RUN

ACEY02  BA    3.0     30-DEC-75

ACEY-DUCEY TWO
------------

DO YOU WISH TO SEE THE INSTRUCTIONS ('YES' OR 'NO')?NO

  YOU NOW HAVE  $ 100 .

HERE ARE YOUR FIRST TWO CARDS...
   FOUR
   TEN

YOUR BET ($)?5

YOUR THIRD CARD IS...
   NINE

YOU WIN!!

YOU NOW HAVE  $ 105 .

HERE ARE YOUR NEXT TWO CARDS...
   NINE
   EIGHT

YOUR BET ($)?0


YOU STILL HAVE  $ 105 .

HERE ARE YOUR NEXT TWO CARDS...
   ACE
   SIX

YOUR BET ($)?^C
READY
```

CALC calculates the values of CLASSIC arithmetic expressions. Use * for multiplication and / for division. Parentheses are allowed.

```
OLD RXA1:CALC

READY
RUN

CALC    BA    3.0     03-FEB-76

YOUR EXPRESSION? 3+4

3+4 = 7

YOUR EXPRESSION?2*5

2*5 = 10

YOUR EXPRESSION? 7*4/3

7*4/3 = 9.33333

YOUR EXPRESSION?^C
READY
```

EASY03 finds the factors of numbers that you enter — you type a number and CLASSIC displays all the numbers that will divide into it evenly.

```
OLD RXA1:EASY03

READY
RUN

EASY03  BA    3.0     30-DEC-75

EASY03
------

THIS PROGRAM WILL FIND THE POSITIVE FACTORS OF ANY NUMBER THAT
YOU ENTER.  AFTER YOU HAVE ENTERED ALL THE NUMBERS THAT YOU
ARE INTERESTED IN, ENTER 'QUIT' TO STOP THE PROGRAM.

YOUR NUMBER?60

THE FACTORS OF 60 ARE:
 1
 2
 3
 4
 5
 6
 10
 12
 15
 20
 30
 60

YOUR NUMBER?^C
READY
```

GUESS you try to guess a number between 1 and 100 that the computer has picked.

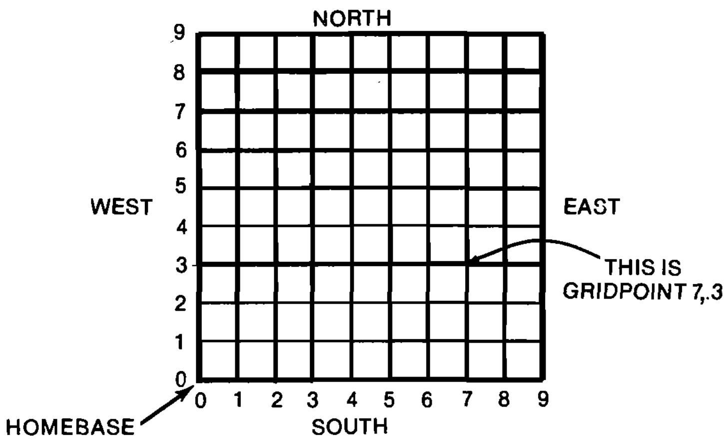
```
OLD RXA1:GUESS

READY
RUN

GUESS   BA    3.0     30-DEC-75

GUESS:  THE NUMBER GUESSING GAME
--------------------------------

PLEASE TYPE YOUR FIRST NAME AND THEN PRESS THE RETURN KEY.

     WHAT IS YOUR FIRST NAME?KATHY

HELLO, KATHY!

I AM THINKING OF A NUMBER BETWEEN  1  AND  100 .
TRY TO GUESS WHAT IT IS.  (PRESS RETURN AFTER EACH GUESS.)

YOUR GUESS?55
   TOO HIGH.  GUESS AGAIN.

YOUR GUESS?^C
READY
```

HMRABI lets you act as governor for the ancient city of Sumeria for a ten-year term of office.

```
OLD RXA1:HMRABI

READY
RUN

HMRABI  BA    3.0     30-DEC-75

TRY YOUR HAND AT GOVERNING ANCIENT SUMERIA
SUCCESSFULLY FOR A 10-YR TERM OF OFFICE.

HAMURABI:  I BEG TO REPORT TO YOU,
IN YEAR 1 , 0 PEOPLE STARVED, 5 CAME TO THE CITY.
POPULATION IS NOW 100
THE CITY NOW OWNS 1000 ACRES.
YOU HARVESTED 3 BUSHELS PER ACRE.
RATS ATE 200 BUSHELS.
YOU NOW HAVE 2800 BUSHELS IN STORE.

LAND IS TRADING AT 20 BUSHELS PER ACRE.
HOW MANY ACRES DO YOU WISH TO BUY?0
HOW MANY ACRES DO YOU WISH TO SELL? 100

HOW MANY BUSHELS DO YOU WISH TO FEED YOUR PEOPLE?  2000

HOW MANY ACRES DO YOU WISH TO PLANT WITH SEED? 900
```

```
HAMURABI:  I BEG TO REPORT TO YOU,
IN YEAR 2 , 0 PEOPLE STARVED, 11 CAME TO THE CITY.
POPULATION IS NOW 111
THE CITY NOW OWNS 900 ACRES.
YOU HARVESTED 1 BUSHELS PER ACRE.
RATS ATE 0 BUSHELS.
YOU NOW HAVE 3250 BUSHELS IN STORE.

LAND IS TRADING AT 25 BUSHELS PER ACRE.
HOW MANY ACRES DO YOU WISH TO BUY?^C
READY
```

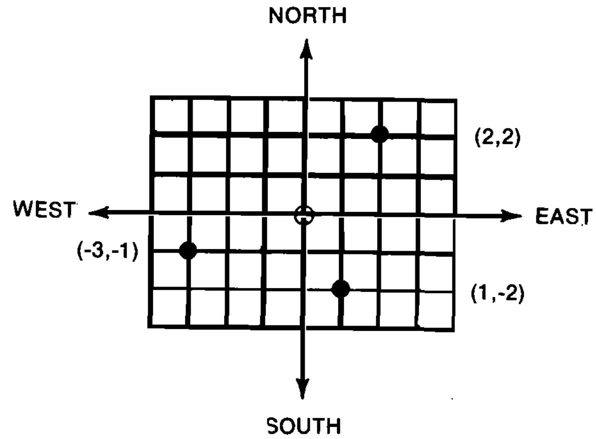
HURKLE hides a Hurkle in a 10 by 10 grid and you guess where he is hiding. The Hurkle's grid looks like this:



You type your guess as two numbers separated by a comma, the first number corresponding to the east-west location and the second to the north-south location.

```
OLD RXA1:HURKLE

READY
RUN

HURKLE  BA     3.0      30-DEC-75

A HURKLE IS HIDING ON A 10 BY 10 GRID. HOMEBASE
ON THE GRID IS POINT  0,0  AND ANY GRIDPOINT IS A
PAIR OF WHOLE NUMBERS SEPARATED BY A COMMA. TRY TO
GUESS THE HURKLE'S GRIDPOINT. YOU GET 5 TRIES.
AFTER EACH TRY, I WILL TELL YOU THE APPROXIMATE
DIRECTION TO GO TO LOOK FOR THE HURKLE.

GUESS # 1 ? 4,4
GO SOUTHEAST

GUESS # 2 ? 2,5
GO SOUTHEAST

GUESS # 3 ? 1,6
GO SOUTHEAST

GUESS # 4 ?^C
READY
```

HURK02 is a more difficult version of HURKLE that uses a grid with both positive and negative locations (a Cartesian coordinate system). The HURK02 grid looks like this:




```
OLD RXA1:HURK02

READY
RUN

HURK02  BA     3.0      30-DEC-75

HURKLE TWO
---------

DO YOU WISH TO SEE THE INSTRUCTIONS ('YES' OR 'NO')?NO

YOUR AVAILABLE OPTIONS ARE NOW 'GO', 'HELP', 'INSTR', 'QUIT', 'SIZE',
AND 'TRIES'.  WHICH WOULD YOU LIKE TO EXERCISE (ENTER A WORD)?GO

  THE HURKLE IS HIDING IN AN 8 BY 8 COORDINATE GRID.  HORIZONTAL
VALUES GO FROM -4 TO 4 AND VERTICAL VALUES GO FROM -4 TO 4 .  FIND
THE HURKLE WITHIN 6 GUESSES!

YOUR FIRST GUESS (ENTER COORDINATES SEPARATED BY A COMMA)?  0,0
GO EAST...

YOUR SECOND GUESS?5,0
   YOUR FIRST COORDINATE IS OUTSIDE OF THE HURKLE'S GRID!  TRY AGAIN...

YOUR SECOND GUESS?4,0
GO WEST...

YOUR THIRD GUESS?2,0
GO WEST...

YOUR FOURTH GUESS? 1,0

HURK!  HURK!  YOU FOUND THE HURKLE IN 4 GUESSES!!

IF YOU'D LIKE TO PLAY AGAIN, PLEASE ENTER THE 'GO' OPTION BELOW.

YOUR AVAILABLE OPTIONS ARE NOW 'GO', 'HELP', 'INSTR', 'QUIT', 'SIZE',
AND 'TRIES'.  WHICH WOULD YOU LIKE TO EXERCISE (ENTER A WORD)?^C
READY
```

MORGAG computes the monthly payments on a mortgage or any other long term loan. You supply the amount of the loan, the annual interest rate, and the number of years that you will be allowed to pay back the loan.

```
OLD RXA1:MORGAG

READY
RUN

MORGAG  BA     3.0      30-DEC-75

COMPUTATION OF MORGAGE PAYMENTS

PLEASE INPUT THE PRINCIPAL (WITHOUT COMMAS)? 29200
INPUT THE ANNUAL INTEREST RATE (IN %)?9
INPUT THE TERM (IN YEARS)?25

PRINCIPAL                         $ 29200
INTEREST RATE                         9 %
TERM                                300   MONTHS
MONTHLY PAYMENT                   $   245.05
```

```
IF YOU WANT THE MONTHLY BREAKDOWN ON THE SCREEN, ENTER 'SCREEN'.
IF YOU WANT IT ON DISK ENTER 'DISK'.
IF YOU DON'T WANT IT AT ALL ENTER 'NO'.
YOUR ENTRY? SCREEN

       OUTSTANDING     INTEREST    PRINCIPAL     TOTAL         TOTAL
MONTH   PRINCIPAL      PAYMENT      PAYMENT     INTEREST     PRINCIPAL

  1      29200          219          26.05       219            26.05
  2      29173.9        218.8        26.25       437.8          52.3
  3      29147.7        218.61       26.44       656.41         78.74
  4      29121.2        218.41       26.64       874.82        105.38
  5      29094.6        218.21       26.84      1093.03        132.22
  6      29067.8        218.01       27.04      1311.04        159.26
  7      29040.7        217.81       27.24      1528.85        186.5
  8      29013.5        217.6        27.45      1746.45        213.95
  9     ^C
READY
```

QUAD03 finds the roots of a quadratic equation. You supply the values of A, B, and C for the equation:

$$A x^2 + B x + C = 0$$

and the computer will tell you what values of x will make the equation true.

```
OLD RXA1:QUAD03

READY
RUN

QUAD03  BA    3.0     30-DEC-75

THIS PROGRAM WILL SOLVE THE QUADRATIC EQUATION IN THE FORM:
AX^2 + BX + C = 0.
AFTER EACH ?, TYPE THE REQUESTED VALUE & PUSH RETURN.

A = ?2
B = ?40
C = ?8
THE ROOTS OF  2 X^2 + 40 X + 8  = 0 ARE:
-0.202042
-19.798

DO YOU WISH TO SOLVE ANOTHER QUADRATIC EQUATION?
ANSWER YES OR NO & PUSH RETURN.?NO

READY
```

SYNONY helps you practice recognizing synonyms by asking you to enter a word having the same meaning as the computer's word. This program presents 10 words and tells you if your answers are correct or incorrect for each one. In addition, the program records the total number of correct and incorrect responses that have been typed for each word.

```
OLD RXA1:SYNONY

READY
RUN

SYNONY  BA    3.0     30-DEC-75

SYNONYMS

IF YOU SEE THE MESSAGE:  EN AT LINE 2020
BELOW, RUN THE PROGRAM 'SYNSET' BY TYPING:
        OLD RXA1:SYNSET
AND THEN:
        RUN

MESSAGE:
NO ERROR MESSAGE

A SYNONYM OF A WORD IS ANOTHER WORD IN THE ENGLISH LANGUAGE
WHICH HAS THE SAME OR VERY NEARLY THE SAME MEANING.

I CHOOSE A WORD -- YOU TYPE A SYNONYM.

WHAT IS A SYNONYM OF FIRST?^C
READY
```

WTDAVG calculates a weighted average for a set of up to 100 numbers. You enter the weights for each number in the set and then you may enter as many sets as you like. This program has several options that you can exercise (such as changing the weights for each grade) which are explained in the instructions.

```
OLD RXA1:WTDAVG

READY
RUN

WTDAVG  BA    3.0     30-DEC-75

WEIGHTED AVERAGING
------------------

DO YOU WISH TO SEE THE INSTRUCTIONS ('YES' OR 'NO')?NO

HOW MANY GRADES DO YOU HAVE FOR EACH STUDENT?3

INPUT YOUR RELATIVE WEIGHTS FOR EACH GRADE BELOW:
     WEIGHT FOR GRADE # 1 ?2
     WEIGHT FOR GRADE # 2 ?1
     WEIGHT FOR GRADE # 3 ?3

INPUT YOUR GRADES FOR STUDENT # 1 BELOW:
     GRADE # 1 ?80
     GRADE # 2 ?75
     GRADE # 3 ?95

THE WEIGHTED AVERAGE OF STUDENT # 1 'S GRADES  =  86.6666


INPUT YOUR GRADES FOR STUDENT # 2 BELOW:
     GRADE # 1 ?^C
READY
```

Chapter 2
Using Classic

WHAT IS CLASSIC?

CLASSIC is a computer system that is made up of three parts: hardware, software, and documentation. The **hardware** is that part of the system that you can see and touch and bump into. The **software** is made up of programs that control how the computer works. (Think of a television set: the set itself is hardware, but the programs that you see and hear are software.) This guide is part of the CLASSIC **documentation** which explains how to use the system. Each of these three parts is described below in more detail.

HARDWARE

The CLASSIC hardware consists of four units (or devices):

(1) the desk,
(2) the keyboard/screen,
(3) the disk drives, and
(4) the central processing unit.

The locations of these units are shown in Figure 2-1.

Desk. Your computer system is housed completely within a movable desk. All the parts needed for CLASSIC to work are put together so that the system may be moved from one classroom to another quickly and easily.

Keyboard/screen. The CLASSIC keyboard and screen are used to "talk" or interact with the computer. When the computer is running, you press keys on the keyboard and those letters or numbers will appear on the screen. The keyboard looks like a standard typewriter; the screen is like a small television. These two devices together are usually called the computer **terminal.**

Disk drives. CLASSIC comes with several flexible disks that can store your work in much the same way that tapes for tape recorders store music. Some of these disks are **pre-recorded** and are needed to make the system work. Other disks are blank, allowing you to store your own work. To be used, the disks must be placed in the disk drives just as records must be placed on a record player before you can listen to them.

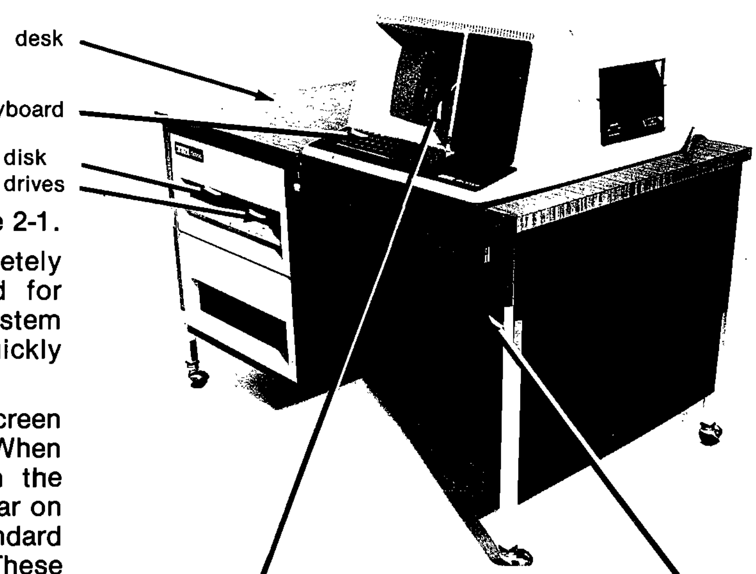


Figure 2-1
CLASSIC Hardware

Central Processing Unit. The "heart" of your CLASSIC system is the central processing unit (CPU) which is hidden at the very back of the desk. The CPU is like the system's motor: it must run for the system to do anything at all. The CPU is sometimes referred to simply as the computer.

Figure 2-2 shows how the CLASSIC hardware units relate to each other. Directions for using each unit are given in the *CLASSIC User's Reference Guide.* Suggestions for keeping the hardware working properly and correcting minor problems are presented in the *CLASSIC Installation and Maintenance Guide.*

SOFTWARE

CLASSIC can run three different types of programs: the monitor program, the editor program, and BASIC language programs.

When you push the white START button, CLASSIC automatically runs the **monitor** program. You can tell when this program is running because it prints a dot (.) or an asterisk (*) when it is waiting for you to type. The lines that you type when the monitor program is running are called **monitor commands.** For example, the line R BASIC typed after the dot is a monitor command. Monitor commands are used to perform certain operations such as copying programs from one disk to another.

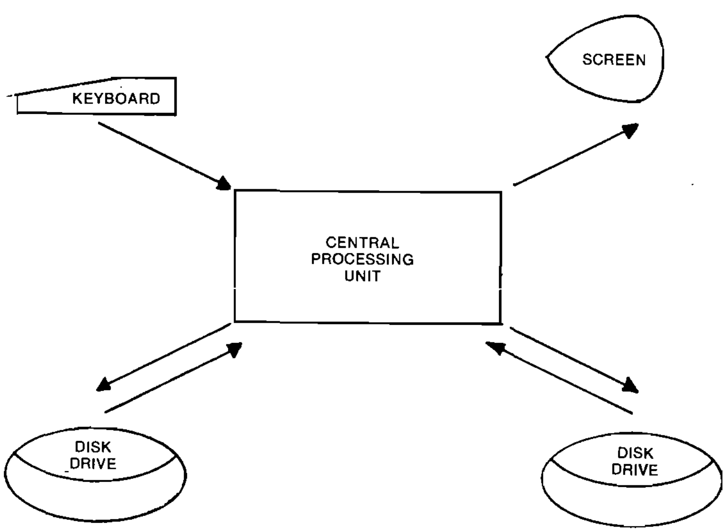


Figure 2-2
Relationships Between CLASSIC Hardware Units

By typing the monitor command R BASIC, you ask CLASSIC to run the **editor** program. The lines you type when the editor program is running are called **editor commands.** For example, the line OLD RXA1:GUESS is an editor command. The editor program does **not** print a dot, but does print the word READY after it completes certain jobs. Editor commands are used to write, change, and run **BASIC language programs.**

BASIC language programs differ from the other types of CLASSIC programs because you can write them. BASIC is a language similar to English, and writing a program in BASIC is like writing directions in English. You may think of a BASIC language program as a recipe that tells the computer how to do a specific job, and each statement line in the program is like a single step in that recipe. To display the statements that make up a BASIC language program stored in the computer's memory on the screen, you can use the editor LIST command.

The following example demonstrates the difference between monitor commands, editor commands, and BASIC language statements. Underlined commands are typed by the user.

```
.R BASIC                                            NOTE 1
NEW OR OLD -OLD RXA1:GUESS                          NOTE 2

READY
LIST 800                                            NOTE 3

GUESS    BA     3.0      03-FEB-76

800 REM    ****    TOO LOW OR TOO HIGH
810 REM
820 PRINT '   TOO ';
830 IF G>N THEN 860
840 PRINT 'LOW';
850 GOTO 870                                        NOTE 4
860 PRINT 'HIGH';
870 PRINT '.  GUESS AGAIN.'
880 PRINT
890 LET K=K+1
900 GOTO 630
910 END

READY
```

NOTES:

(1) R BASIC is a **monitor command** that tells CLASSIC to run the editor program. Notice the dot that precedes this command. The dot was printed by the *monitor program*, not typed by the user.

(2) OLD RXA1:GUESS is an **editor command** that tells CLASSIC to find the program called GUESS on disk drive 1 (RXA1) and put it into the computer's memory. Notice that READY is printed when this operation is completed.

(3) LIST 800 is an **editor command** that tells CLASSIC to display the program stored in its memory on the screen, beginning with line 800.

(4) These are the **BASIC language statements** that make up part of the program GUESS. Note that each begins with a **line number** and is made up of simple English words or mathematical expressions.

DOCUMENTATION

The CLASSIC documentation is made up of three Guides:

(1) *CLASSIC Installation and Maintenance Guide*
(2) *The CLASSIC Primer: A Self-Teaching Guide*
(3) *CLASSIC User's Reference Guide*

These guides contain all the information that you will need to work with CLASSIC, from installing it to writing BASIC language computer programs to correcting minor problems.

CLASSIC Installation and Maintenance Guide. The CLASSIC system is designed so that it can be installed by anyone who carefully reads and follows the directions. The installation involves uncrating the system, connecting its units, testing its operation, and copying the BASIC system disk. The *CLASSIC Installation and Maintenance Guide* provides step-by-step instructions for each of these four processes and contains a complete maintenance section to help you keep your CLASSIC in top working order and direct you in correcting minor problems.

The CLASSIC Primer: A Self-Teaching Guide. The CLASSIC Primer will help you teach yourself how to work with CLASSIC. The first few chapters will lead you through the use of the CLASSIC software, and the last chapter will help you discover some of the many ways to use CLASSIC and find further information on computer uses in instruction. (If you have the optional FORTRAN IV software but have never used a computer before, it is recommended that you teach yourself BASIC before you try to learn FORTRAN.)

CLASSIC User's Reference Guide. Once you have learned to use CLASSIC, you will often need a reference to help you remember rules and the meanings of error messages. This information is collected in the CLASSIC User's Reference Guide.

If you have the optional FORTRAN IV software, you will also need the *OS/8 Handbook* (order number DEC-08-OSHBA-A-D). Pages 1-78 to 1-92 of the *OS/8 Handbook* explain how to create a FORTRAN program file with the Symbolic Editor. Pages 8-1 to 8-64 describe how a FORTRAN program is compiled, loaded, and executed, and pages 8-65 to 8-124 discuss the various statements that make up the FORTRAN IV language.

USING THE CLASSIC SOFTWARE

As you learn to work with CLASSIC, you will make mistakes. Some of your mistakes will be minor and can be easily corrected. Others will be major and may even destroy part of the CLASSIC software. To correct these major errors, you will need a back-up or duplicate copy of your system disks. Therefore,

BEFORE YOU DO ANY WORK ON YOUR SYSTEM, MAKE SURE THAT THE PERSON IN CHARGE OF YOUR CLASSIC SYSTEM HAS BACK-UP COPIES OF ALL THE DISKS THAT YOU WILL USE.

TYPING RULES USED IN THIS GUIDE

Two conventions will be used throughout this Guide to indicate what CLASSIC will display and what you should type.

First, everything that you must enter (type in) through the keyboard will be underlined. Anything that is **not** underlined is displayed by CLASSIC. Look at the following example:

.<u>DATE</u>
NONE

In this example, CLASSIC displays the first dot, you type "DATE" (and then press the RETURN key), and the system displays "NONE" and the second dot.

Second, "0" will be used to stand for the **number** "zero" and "O" for the **letter** "oh". You should also note that there is a "1" key at the upper left-hand corner of your keyboard which must be used to type the number "one". CLASSIC does not recognize lower case letters, so neither the lower case "L" ("l") nor the upper case "I" can be used for the number "one" as might be done on a standard typewriter.

Chapter 3 Beginning Basic Programming

UNDERSTANDING WHAT TO DO

In Chapter 1 you learned how to start CLASSIC and run a program. Chapter 2 explained the difference between the monitor program, the editor program, and BASIC language programs. This chapter will help you teach yourself about CLASSIC by writing programs in the BASIC language and using various monitor and editor commands.

Each section in Chapter 3 contains exercises to help you understand how CLASSIC works. Suggested answers to these exercises are given in Appendix C. For some exercises, however, there may be more than one correct answer, especially when you are asked to write your own computer programs.

SECTION 3-A

MAKING CALCULATIONS

ENTERING BASIC PROGRAMS

When you typed R BASIC in Chapter 1, CLASSIC set aside a certain area of its memory for you to use as a **workspace.** The workspace is used to write and run BASIC language programs. When you typed OLD RXA1:GUESS, you told CLASSIC to read the program GUESS into your workspace from the disk that it knows as RXA1. (RXA1 always refers to the disk in the right-hand disk drive.) To tell CLASSIC that you want to enter a **new** program into the workspace from the keyboard, you could use the editor NEW command. For example, you might type:

.R BASIC
NEW OR OLD—NEW PROG1
READY

(Remember that lines that are not underlined are typed by the computer, and lines that are underlined must be typed by you and ended by pressing the RETURN key.) The command:

NEW PROG1

tells CLASSIC that you want to write or **enter** a new program called PROG1 into the workspace.

If you then type LIST (and push RETURN), CLASSIC will print:

PROG1 BA 3.0 THIS IS THE **HEADER** OF YOUR PROGRAM.
READY

LIST is an editor command just like OLD, RUN, and NEW. It tells CLASSIC to list the program in your workspace. If you did not use the OLD command to read a program into the workspace and have not yet put a new program into it, your workspace is empty and only the **header** will be displayed. The header consists of:

(1) the **name** of your program (PROG1),
(2) its **extension** (a two-letter code indicating its type, usually BA for BASIC language programs), and
(3) the **version number** of the CLASSIC software (3.0).

When the READY message appears, you may begin entering a BASIC language program into the workspace. This is done simply by typing BASIC language statements at the keyboard. For example, you might type:

10 PRINT 7
99 END

This program will then be in your workspace. The program consists of two **statements,** a PRINT statement and an END statement.

The END statement must always be the last statement in your program.

Notice that each statement begins with a **line number.** You may enter statements in any order, but CLASSIC will automatically put them in order by their line numbers.

If you type LIST after this program has been entered, the new contents of the workspace will be displayed.

```
LIST
PROG1     BA     3.0

10 PRINT 7
99 END
READY
```

To run this program, you must type RUN (and press RETURN). In a few seconds, CLASSIC should print:

```
RUN
PROG1     BA     3.0

7

READY
```

If it does not, your program contains an **error** and CLASSIC will print an **error message.** At this stage, correct your errors simply by retyping your program and RUNning it again. Error messages will be explained later.

SCRATCH is another editor command. It tells CLASSIC to erase the program in your workspace. If you enter the SCRATCH command, your workspace will be empty, just as it was after the NEW command. You might think of your workspace as a chalkboard that can be erased by typing SCRATCH. The editor SCRATCH command may be abbreviated to SC.

Exercise 1. This exercise will help clarify the steps that you must follow to enter and run a BASIC language program.

Start CLASSIC as you did in Steps 1 through 8 of Chapter 1. Then type the lines shown below. If you make a mistake, simply retype the line.

.R BASIC	Tell CLASSIC to run the editor program.
NEW OR OLD—NEW FIRST	Tell CLASSIC that you are about to enter a new program called FIRST.
READY	
10 PRINT 3+4 *99 END*	Type these lines. This new program contains two statements. END is the last statement because it has the highest line number.
RUN	Tell CLASSIC to RUN the program.
FIRST BA 3.0	This is the program header.
7	The result is 7 since 3+4=7.
READY	
20 PRINT 3-4 *30 PRINT 3*4* *40 PRINT 3/4*	Add these three statements to your program by typing them with the line numbers 20, 30, and 40.
LIST	Tell CLASSIC to LIST the program in its workspace.
FIRST BA 3.0	
10 PRINT 3+4 *20 PRINT 3-4* *30 PRINT 3*4* *40 PRINT 3/4* *99 END*	Note that CLASSIC puts the statements in order by the line numbers.
RUN	Now RUN the program in your workspace.
FIRST BA 3.0	
7 *-1* *12* *0.75*	These are the four results, one for each of the first four statements in your program.
READY	

Look at your program more carefully. Note the symbol that is used to perform each of the four arithmetic operations.

Operation	**Symbol**
Addition	+
Subtraction	-
Multiplication	*
Division	/

Exercise 2. SCRATCH your workspace and enter the following program:

```
10 PRINT 12+3
20 PRINT 12-3
30 PRINT 12*3
40 PRINT 12/3
99 END
```

Before you RUN this program, write down what you think the computer will print. Then RUN the program to check yourself.

Exercise 3. Write original programs using the PRINT and END statements to make other calculations. Be sure to SCRATCH your workspace between each program and include the END statement as the last statement in your program.

When you write your own programs, you may use any whole numbers (integers) between 1 and 99999 as line numbers.

CLASSIC allows line numbers from 1 to 99999.

However, instead of numbering statements with consecutive numbers (1, 2, 3, etc.) use 10, 20, 30, and so on. This gives you room to insert a new statement between two old statements. For example, if you had already entered a program using 10, 20, 30, 40, and 99 as line numbers, you could insert a statement between statement 20 and statement 30 by using 25 as the line number of the new statement.

WRITING NUMERICAL EXPRESSIONS

So far, you have used the PRINT statement in the following form:

line number PRINT numerical expression

For example:

10 PRINT 3+4

line number

PRINT

numerical expression

A PRINT statement in this form tells the computer to calculate the **value** (simplest form) of the numerical expression and print the result on the screen.

A numerical expression can contain more than one operation. For example, the program:

```
10 PRINT 3+4+5
99 END
```

will print the number 12 on the screen. CLASSIC usually prints the value of a numerical expression as a decimal number. The following table shows the values that CLASSIC will print for certain numerical expressions.

Expression	Value Printed	Remarks
3.14	3.14	
-123	-123	
2+3+4	9	2+3+4=5+4=9
2*3/4	1.5	2*3/4=6/4=1.5
1/2+3	3.5	1/2+3=.5+3=3.5
2+3/4	2.75	2+3/4=2+.75=2.75
1/(2+3)	0.2	1/(2+3)=1/5=.2
(2+3)/4	1.25	(2+3)/4=5/4=1.25
1/3	0.333333	Value truncated to six digits
100/3	33.3333	

The table above illustrates each of the following rules:

(1) Arithmetic operations are done in order from left to right.

(2) **All** multiplications and divisions are done before **any** additions or subtractions. For example, to evaluate the numerical expression:

4+24/3*2-5

CLASSIC:	4+24/3*2-5
(a) divides 24 by 3 to get 8,	4+ 8 *2-5
(b) multiplies 8 times 2 to get 16,	4+ 16 -5
(c) adds 4 to 16 to get 20, and then	20 -5
(d) subtracts 5 from 20 to get 15.	15

(3) Parentheses can be used to change the order in which operations are done: all calculations within parentheses are done before those outside parentheses. For example, to evaluate the numerical expression:

((6+14)/2-6)*3

CLASSIC:	((6+14)/2-6)*3
(a) adds 6 to 14 to get 20,	(20 /2-6)*3
(b) divides 20 by 2 to get 10,	(10 -6)*3
(c) subtracts 6 from 10 to get 4, and then	4 *3
(d) multiplies 4 times 3 to get 12.	12

There is a special program on the BASIC Program Demonstration disk that you can use to experiment with numerical expressions. This program is called CALC and evaluates numerical expressions. A sample RUN of this program is shown on the next page. Note that "QUIT" may be used to terminate this program. (CTRL/C will also work.)

Exercise 4. RUN program CALC from the BASIC Program Demonstration disk and experiment with various combinations of the four operations and parentheses. If you make a mistake that causes CLASSIC to end the program and print an error message, simply type RUN again after the READY message appears and reenter your expression.

Sample Run of CALC:

```
.R BASIC
NEW OR OLD--OLD RXA1:CALC

READY
RUN

CALC     BA     3.0

YOUR EXPRESSION?2*3+4

2*3+4  =  10

YOUR EXPRESSION?2+3*4

2+3*4  =  14

YOUR EXPRESSION?1/(2+3)

1/(2+3)  =  0.2

YOUR EXPRESSION?(2+3)/4

(2+3)/4  =  1.25

YOUR EXPRESSION?1/3

1/3  =  0.333333

YOUR EXPRESSION?2/3

2/3  =  0.666666

YOUR EXPRESSION?QUIT

READY
```

LOOKING BACK

You now know five editor commands:

LIST	lists the program in the workspace
NEW	enters a new program into the workspace from the keyboard
OLD	reads a program into the workspace from a disk
RUN	runs the program in the workspace
SCRATCH	erases the program in the workspace

You also know two BASIC language statements:

PRINT	prints the value of a numerical expression on the screen
END	signals the end of a BASIC program

In addition, you should remember the following rules:

(1) BASIC programs are made up of **statements.**

(2) Each BASIC language statement begins with a **line number.**

(3) A line number may be any whole number between 1 and 99999.

(4) The last statement in a BASIC program must be an END statement.

(5) When evaluating a numerical expression, CLASSIC calculates values inside parentheses first, then does all multiplications and divisions from left to right, and finally does all additions and subtractions from left to right.

(6) If an expression contains parentheses within parentheses, expressions are evaluated from the innermost parentheses out.

(7) Program statements with mistakes can be corrected by simply retyping them.

(8) Additional statements can be inserted into an existing program by using the appropriate line numbers.

The next section will talk about more things that you can do with the PRINT statement.

SECTION 3-B

PRINTING LARGER NUMBERS AND WORDS

USING COMMAS AND SEMICOLONS

In the previous section you used the PRINT statement in the form:

line number PRINT numerical expression

A more general form of the PRINT statement is shown below:

line number PRINT list of expressions

For example:

*10 PRINT 3+4,3-4,3*4,3/4*

line number
PRINT
list of expressions

Note that the expressions in this PRINT statement are separated by commas. The program:

```
10 PRINT 3+4,3-4,3*4,3/4
99 END
```

will cause the computer to print the following results:

```
RUNNH
7       -1       12       0.75
READY
```

RUNNH tells the computer to RUN the program in the workspace but without printing the header (NH stands for No Header).

The computer prints a result for each expression in the PRINT statement. Since the statement contained four expressions, four results were printed.

The results PRINTed by a program are called the program *output*.

When commas are used, CLASSIC will print up to five results on each line. If there are more than five expressions in the PRINT statement, additional results are automatically printed on the next line.

For example, the statement:

```
10 PRINT 3+4,3-4,3*4,3/4,3*4*5,3*4/5,3/4*5
```

will cause the computer to print the following results:

```
RUNNH
7        -1        12        0.75        60
2.4      3.75
```

You can think of a line on your screen as being divided into five **print zones**, each 14 spaces wide.

A comma in a PRINT statement tells CLASSIC to move to the next print zone before printing the next result.

If you use a semicolon (;) instead of a comma to separate expressions, the results will be packed more closely together:

```
10 PRINT 3+4;3-4;3*4;3/4      Note the semicolons (;).
99 END
RUNNH
7 -1 12 0.75                  The results are "packed" more
                              closely together than if you
READY                         had used commas.
```

LISTNH tells the computer to LIST the program in the workspace but without printing the header (NH stands for No Header).

```
LISTNH
10 PRINT 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23;24
99 END
```

When you use semicolons to separate expressions, the computer will print up to 24 results per line. The actual number, however, depends on the number of digits that it must print. For example,

```
READY
RUNNH
 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
 21  22  23  24

READY
```

The first 20 results were printed on the first line; the 21st through 24th on the second line.

A semicolon in a PRINT statement tells CLASSIC to print the next result without moving to the next print zone.

Whenever CLASSIC prints a number it uses the following format:

sNb

where: s is the sign of the number ("-" for negative and a blank for positive)
N is the number (up to six digits long)
b is a blank

Thus, at least 3 spaces are needed to print each number. Since output lines (lines PRINTed by programs) may be up to 72 spaces long and 72/3 = 24, up to 24 results may be printed on each line. This format also explains the blank space at the beginning of the output line in the preceding program and the two spaces between each number: each number is preceded by a blank that represents the sign of the number (all positive in this case) and followed by a blank. The next example demonstrates this more clearly:

```
LISTNH
10 PRINT 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23;24
20 PRINT -1;-2;-3;-4;-5;-6;-7;-8;-9;-10;-11;-12;-13;-14;-15;-16;-17;-18;
30 PRINT -19;-20;-21;-22;-23;-24
99 END

READY
RUNNH
 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
 21  22  23  24
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20
-21 -22 -23 -24

READY
```

Using positive and negative numbers, you can more easily see the sign-number-blank format (sNb). Note that if a semicolon or comma ends a PRINT statement (see line 20), the output of the next PRINT statement continues on the same line.

Remember these things:

- A PRINT statement can contain more than one expression.
- One result is printed for **each** expression in a PRINT statement.
- If a PRINT statement contains more than one expression, the expressions must be separated by a comma (,) or semicolon (;).
- If commas are used for spacing, up to five results per line are printed. If semicolons are used, the results are "packed" more closely together. The actual spacing depends on the size of the numbers involved.
- RUNNH is an editor command that tells CLASSIC to run the program in the workspace without printing the program header.
- The results PRINTed by a program are called the program **output.**

Exercise 5. Write a program to produce the following results. Use commas and semicolons to adjust spacing and make your program as short as possible. Test your program on CLASSIC.

```
RUNNH
 11  22           33  44
 0.333333         0.666666  1   1.33333
 0.166667  0.833333  0.125  0.875
-1              -2              -3              -4

READY
```

Exercise 6. Write a program to produce the following results. Hint: you can have more than one punctuation mark between two numbers.

```
RUNNH
                          0.5
              0.25                      0.75
 0
             -0.25                     -0.75
                         -0.5

READY
```

STRINGS (ALPHANUMERICAL EXPRESSIONS)

So far, you have printed only numerical expressions. The PRINT statement in the following program directs the computer to print a **string (alphanumerical expression).**

```
LISTNH
10 PRINT "STRINGS ARE MADE UP OF LETTERS AND NUMBERS."
99 END

READY
RUNNH
STRINGS ARE MADE UP OF LETTERS AND NUMBERS.

READY
```

The string is enclosed in quotes. While a numerical expression may contain only the digits 0-9 and signs for arithmetic operations, a string may contain any printing character on the keyboard except the backslash (\ ) and the underscore (__).

The next example illustrates the difference between strings and numerical expressions:

```
LISTNH
10 PRINT "14.6 * 13.8 ="; 14.6 * 13.8
20 PRINT 3.61 + 8.72; "IS THE SUM OF 3.61 + 8.72"
99 END

READY
RUNNH
14.6 * 13.8 = 201.48
 12.33 IS THE SUM OF 3.61 + 8.72

READY
```

In this example, the strings are:

"14.6 * 13.8 ="

"IS THE SUM OF 3.61 + 8.72"

The numerical expressions are:

14.6 * 13.8

3.61 + 8.72

Exercise 7. On a separate piece of paper, write down what the computer will print when the following program is run. Then run it on the computer to check your answer.

```
10 PRINT "COMPUTERS DO ARITHMETIC LIKE THIS:"
20 PRINT "3+4-5 ="; 3+4-5, "3+4+5 ="; 3+4+5, "3+4/5 ="; 3+4/5
30 PRINT "3-4+5 ="; 3-4+5, "3-4*5 ="; 3-4*5, "3-4/5 ="; 3-4/5
40 PRINT "3*4+5 ="; 3*4+5, "3*4-5 ="; 3*4-5, "3*4/5 ="; 3*4/5
50 PRINT "3/4+5 ="; 3/4+5, "3/4-5 ="; 3/4-5, "3/4*5 ="; 3/4*5
60 PRINT "6+5-4*3/2 ="; 6+5-4*3/2
99 END
```

Except for certain special characters ("\" and "__"), anything enclosed in quotation marks in a PRINT statement is printed exactly as it appears. No arithmetic is performed.

EXPONENTS — RAISING A NUMBER TO A POWER

A number is "raised to a power" by multiplying it by itself. For example, "2 raised to the power of 3" is evaluated (computed) by multiplying 2 times itself three times:

$$2^3 = 2 \times 2 \times 2 = 8$$

In this expression 3 is the **exponent** of 2.

CLASSIC uses the circumflex (^) to indicate the operation of **exponentiation** — raising a number to a power. (The circumflex is on the top row of keys above the 6.) For example, 2^3 would be typed as 2^3. The following program illustrates exponentiation on CLASSIC:

```
LISTNH
10 PRINT "5^2 = 5*5 ="; 5^2
20 PRINT "2^3 = 2*2*2 ="; 2^3
30 PRINT "3^4 = 3*3*3*3 ="; 3^4
99 END

READY
RUNNH
5^2 = 5*5 = 25
2^3 = 2*2*2 = 8
3^4 = 3*3*3*3 = 81

READY
```

Here are some examples showing the values of numerical expressions in which the ∧ is used.

Expression	Value	Remarks
2∧5	32	2∧5 = 2*2*2*2*2* = 32
3∧2 + 4∧2	25	3∧2 + 4∧2 = 9 + 16 = 25
(2+3)∧4	625	(2+3)∧4 = 5∧4 = 5*5*5*5 = 625

When an expression contains both exponentiation and other arithmetic functions, the exponentiation is always done first. This order may, however, be changed by using parentheses. For example, to evaluate the expression:

(7-5) ∧ 4*(8+2)

CLASSIC:

(7-5) ∧ 4*(8+2)

(a) subtracts 5 from 7 to get 2. — 2 ∧ 4*(8+2)
(b) adds 8 to 2 to get 10, — 2 ∧ 4* 10
(c) raises 2 to the 4th power to get 16, — 16 * 10
(d) and multiplies 16 times 10 to get 160. — 160

Exercise 8. Write your own programs or use the CALC program on the BASIC Program Demonstration disk to experiment with exponents by finding the values of the following expressions:

(1) 12 ∧ (4/2) (6) 10^{10-6}
(2) 5^5 (7) (2+6) ∧ (4-2)
(3) 3/4 ∧ 2 (8) 7 ∧ 1
(4) (3/4) ∧ 2 (9) 7 ∧ 0
(5) 3/(4 ∧ 2) (10) 0 ∧ 8

FLOATING -POINT NOTATION

CLASSIC displays very large and very small numbers in **floating-point notation:**

```
LISTNH
10 PRINT 10
20 PRINT 100
30 PRINT 1000
40 PRINT 10000
50 PRINT 100000
60 PRINT 1000000
70 PRINT 10000000
99 END
```

In the program, each number is expressed in **standard** or **common** notation.

```
READY
RUNNH
 10
 100
 1000
 10000
 100000
```

These numbers are printed in standard notation, exactly as they are written in the PRINT statements.

```
 .100000E+007
 .100000E+008
READY
```

But these are printed in **floating-point notation.**

When you read numbers written in floating-point notation, substitute the words "times ten to the power of" for the letter "E".

If a number is larger than 999999, it will be printed in floating-point notation.

The following examples show the same numbers expressed in standard notation, scientific notation, and floating-point notation.

Standard Notation	Scientific Notation	Floating-Point
1000000	1×10^6	.100000E+007
10000000	1×10^7	.100000E+008
100000000	1×10^8	.100000E+009
1000000000000	1×10^{12}	.100000E+013

Look what happens when CLASSIC handles small numbers:

```
LISTNH
10 PRINT .1
20 PRINT .01
30 PRINT .001
40 PRINT .0001
50 PRINT .00001
60 PRINT .000001
70 PRINT .0000001
80 PRINT .00000001
90 PRINT .000000001
99 END
```

These numbers have more than six decimal places ...

```
READY
RUNNH
 0.1
 0.0099999
 0.001
 0.0001
 0.00001
 0.000001
 .999999E-007
 .999999E-008
 .999999E-009
READY
```

...so they are printed in floating-point notation.

But now there is a new problem: why did CLASSIC print 0.0099999 for line 20 instead of 0.01? And why did it print all those 9's in the last three lines? The answer is that when CLASSIC handles numbers less than 1, it sometimes converts from standard notation to floating-point notation as shown in the following table.

Standard Notation	Floating-Point		
.1	0.1	or	0.0999999
.01	0.01	or	0.0099999
.001	0.001	or	0.0009999
.0001	0.0001	or	0.0000999
.00001	0.00001	or	0.0000099
.000001	0.000001	or	0.0000009
.0000001	.100000E-006	or	.999999E-007
.00000001	.100000E-007	or	.999999E-008
.000000001	.100000E-008	or	.999999E-009

In general, floating-point notation is used for numbers that require more than 6 digits in standard notation. However, the number after the letter E must be less than 617 and greater than -617.

You may use floating-point notation whenever you wish to specify numbers. If the number can be written in standard notation, a conversion will be made before it is displayed. The following program demonstrates this:

```
LISTNH
10 PRINT 3E4+5E4
20 PRINT 3E4-5E4
30 PRINT 3E4*5E4
40 PRINT 3E4/5E4
99 END

READY
RUNNH
 80000
-20000
 .150000E+010
 0.6

READY
```

Exercise 9. Write your own programs or use the CALC program to experiment with exponentiation and floating-point notation before you go on. A sample run of the CALC program demonstrating these features is shown at the right. Note the ways that floating-point numbers may be entered. "E" is considered to be part of the number just like the digits 0-9 and the signs + or -.

LOOKING BACK

In this section you have looked at ways to use the PRINT statement with large numbers and words.

Sample Run of Program CALC:

```
.R BASIC
NEW OR OLD--OLD RXA1:CALC

READY
RUNNH

YOUR EXPRESSION?10000000000

10000000000 = .999999E+010

YOUR EXPRESSION?.0000000001

.0000000001 = .999999E-010

YOUR EXPRESSION?4E-6

4E-6 = 0.000004

YOUR EXPRESSION?4E-12

4E-12 = .399999E-011

YOUR EXPRESSION?16^16

16^16 = .184467E+020

YOUR EXPRESSION?35*4^36

35*4^36 = .165282E+024

YOUR EXPRESSION?2*3^2

2*3^2 = 18

YOUR EXPRESSION?2*2*2*2*2*2*2*2*2*2

2*2*2*2*2*2*2*2*2*2 = 1024

YOUR EXPRESSION?2^2^2^2^2^2^2^2^2^2

2^2^2^2^2^2^2^2^2^2 = .134077E+155

YOUR EXPRESSION?(2+3)-4*5/6^7

(2+3)-4*5/6^7 = 4.99993

YOUR EXPRESSION?-.3*2E456

-.3*2E456 =-.599980E+456

YOUR EXPRESSION?3*4E26-31E450

3*4E26-31E450 =-.309990E+452

YOUR EXPRESSION?QUIT


READY
```

Remember these things:

- A line on the CLASSIC screen is divided into five print zones, each 14 spaces wide.
- A comma in a PRINT statement tells CLASSIC to move to the next print zone before printing the next result.
- A semicolon in a PRINT statement tells CLASSIC to print the next result without moving to the next print zone.
- Strings (alphanumerical expressions) can be made up of any characters on the keyboard except for the backslash (\) and the underscore (_).
- The circumflex (^) is used to indicate exponentiation.
- If a number is larger than 999999 or smaller than .000001, it will be printed in floating-point notation.
- The largest number that CLASSIC can work with is $1x10^{617}$. The smallest is $1x10^{-617}$.

SECTION 3-C

PRINTING VARIABLE RESULTS

The programs that you wrote for the previous sections always printed out the same results each time you ran them. If you wanted to solve a different problem, you had to write a different program. This section will show you how to make a single program print different results.

USING VARIABLES

In mathematics, variables are used to represent unknown numbers. For example, you have probably seen the equation:

$$A = \pi r^2$$

that is used to represent the area of a circle. This equation has two variables, "A" and "r". "π" is a constant, approximately equal to 3.14.

In BASIC, there are several ways to represent variables. One way is to use capital letters. Each capital letter refers to a distinct location in the computer's memory. It may help you to think of part of the computer's memory as containing a set of 26 boxes, labelled A through Z, like this:

A ☐	H ☐	O ☐	V ☐
B ☐	I ☐	P ☐	W ☐
C ☐	J ☐	Q ☐	X ☐
D ☐	K ☐	R ☐	Y ☐
E ☐	L ☐	S ☐	Z ☐
F ☐	M ☐	T ☐	
G ☐	N ☐	U ☐	

Each location can hold one number at any time. The current number in a location is known as the **value** of the variable corresponding to that location. Before a program is run, the values of all numeric variables are 0.

The following example shows how to assign a value to a variable in a BASIC program:

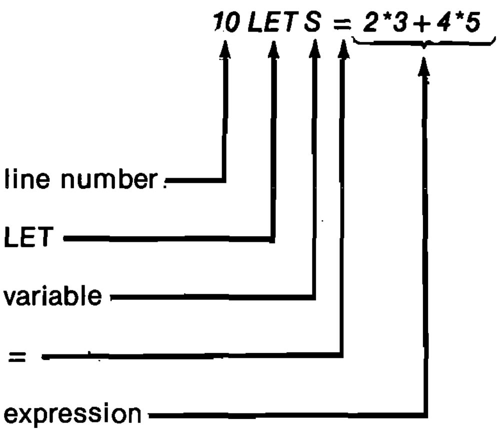
```
10 LET A=3   Assign the value 3 to the variable A.
20 PRINT A   Print the value of A.
99 END
RUNNH
3            The value of A is 3.
READY
```

In its simplest form, the LET statement assigns values of constants to specific locations in the computer's memory.

A more general form of the LET statement is shown below:

line number LET variable = expression

For example:



The following program demonstrates a simple use of variables to evaluate expressions:

```
LISTNH
10 LET A=3
20 LET B=4
30 LET C=3+4
40 LET D=3-4
50 LET E=3*4
60 LET F=3/4
70 LET G=3^4
80 PRINT A;B;C;D;E;F;G
99 END

READY
RUNNH
 3  4  7 -1  12  0.75  81

READY
```

The LET statement tells the computer to calculate the value of the expression to the right of the "=" symbol and assign this value to the variable that appears to the left of the "=" symbol.

The value assigned to a variable in a LET statement replaces any previous value of that variable. For example, look at the following program:

```
LISTNH
10 LET A=1
25 PRINT A
20 LET A=2
25 PRINT A
30 LET A=3
35 PRINT A
99 END

READY
RUNNH
 1
 2
 3

READY
```

Each time A is printed (lines 15, 25, and 35), a different result is displayed (first "1", then "2", then "3"). The following table shows why this occurred by tracing the value of A as each statement is executed.

Statement	Value of A	Remarks
10 LET A = 1	1	Assign the value 1 to A.
15 PRINT A	1	Print the current value of A.
20 LET A = 2	2	Assign the value 2 to A.
25 PRINT A	2	Print the current value of A.
30 LET A = 3	3	Assign the value 3 to A.
35 PRINT A	3	Print the current value of A.
99 END	3	

Exercise 10. What values will be printed by the following programs? Write your answers on a piece of paper and then check yourself by running the programs on CLASSIC.

```
10 LET X=3
20 LET X=5
30 LET X=7
40 PRINT X
99 END
```

```
10 LET X=3
20 LET Y=5
30 LET Z=7
40 PRINT X,Y,Z
99 END
```

VARIABLE EXPRESSIONS

A **variable expression** is an expression that contains one or more variables. For example, the following are variable expressions:

A	-C
A - B	A*(B+C)
2*X	A/B+C/D
P/Q	3.14*R∧2

The computer evaluates a variable expression by assigning values to its variable or variables and carrying out the indicated operations.

For example, A*B is a variable expression with variables A and B. If A = 3 and B = 4, then the value of A*B is 12. But if A = -7 and B = 5, then the value of A*B is -35.

Here are some more examples:

Variable Expression	Value(s) of Variable(s)	Value of Expression
A	A=3	3
	A=-123	-123
A-B	A=12 and B=7	5
	A=3 and B=4	-1
2*X	X=3.14	6.28
	X=-6	-12
P/Q	P=35 and Q=5	7
	P=2 and Q=3	0.666666

Variable expression	Value(s) of Variable(s)	Value of Expression
-C	C=8	-8
	C=0	0
	C=-12	12
A*(B+C)	A=3, B=4, C=5	27
3.14*R∧2	R=3	28.26

Each of the following programs directs the computer to evaluate a variable expression and print the result.

```
LISTNH
10 LET A=3
20 LET B=4
30 PRINT A+B
99 END

READY
RUNNH
 7

READY
```

```
LISTNH
10 LET A=3
20 LET B=4
30 PRINT A*B
99 END

READY
RUNNH
 12

READY
```

Exercise 11. What values will be printed by the following programs? Write down your answer on a separate piece of paper and check yourself by running the program on CLASSIC.

```
10 LET A=3
20 LET B=4
30 PRINT A-B
99 END
```

```
10 LET A=3
20 LET B=4
30 PRINT A-B
99 END
```

THE INPUT STATEMENT

At the beginning of this section you saw the equation:

$$A = \pi r^2$$

which can be used to calculate the area, "A", of a circle with radius "r". To use CLASSIC to calculate the area of a circle, you can translate πr^2 to the BASIC statement:

*20 PRINT 3.14*R∧2* (*π is approximately equal to 3.14*)

The following discussion shows how you can use a variation of this statement to find the areas of circles with different radii:

```
LISTNH
10 LET R=2
15 PRINT 'RADIUS', 'AREA'
20 PRINT R, 3.14*R^2
99 END

READY
```

Here is the program. It will work for R=2.

```
RUNNH
RADIUS        AREA
 2             12.56

READY
```

Run it.

For R=2, the area is 12.56.

Do **NOT** clear the workspace. Instead enter a **new** statement 10 and keep the other three statements.

```
10 LET R=3

RUNNH
RADIUS        AREA
 3             28.26

READY
```

For R=3, A=28.26.

```
10 LET R=8
```

Change statement 10 again.

```
RUNNH
RADIUS        AREA
 8             200.96

READY
```

And run the program.

For R=8, A=200.96.

You can reduce the amount of work required to find the three areas by using the INPUT statement. Here is a program that uses an INPUT statement to permit input of a value of R:

```
LISTNH
10 INPUT R
15 PRINT 'RADIUS', 'AREA'
20 PRINT R, 3.14*R^2
99 END

READY
RUNNH
?
```

The computer types a question mark and waits.

The question mark indicates that CLASSIC is waiting for you to enter *data*. Data consists of numbers and/or strings that are manipulated when a program is executed.

If you enter 2 as your data and press the RETURN key, CLASSIC will print:

```
RADIUS        AREA
 2             12.56

READY
```

meaning that a circle with a radius of 2 has an area of 12.56.

The next example demonstrates a run of the above program for R=2, R=3, and R=8. (Note how the PRINT statement at line 5 is used to tell the user the type of entry that should be made.)

```
LISTNH
5 PRINT 'RADIUS';
10 INPUT R
15 PRINT 'RADIUS', 'AREA'
20 PRINT R, 3.14*R^2
99 END

READY
```

Run the program.

```
RUNNH
RADIUS?2
```

Enter 2 and press RETURN.

```
RADIUS         AREA
 2              12.56
```

For R=2, A=12.56.

```
READY

RUNNH
RADIUS?3
RADIUS         AREA
 3              28.26

READY
```

Run the program again.
Enter 3 and press RETURN.
For R=3, A=28.26.

```
RUNNH
RADIUS?8
RADIUS         AREA
 8              200.96

READY
```

Run the program again.
Enter 8 and press RETURN.
For R=8, A=200.96.

Numbers may be entered in standard notation as shown above or in floating-point notation:

```
LISTNH
5 PRINT 'RADIUS';
10 INPUT R
15 PRINT 'RADIUS', 'AREA'
20 PRINT R, 3.14*R^2
99 END

READY

RUNNH
RADIUS?3.6E4
RADIUS         AREA
 36000          .406944E+010

READY
```

The general form of the INPUT statement is:

line number   INPUT   list of variables

For example:

10 INPUT A,B,C

line number

INPUT

list of variables

Note that only the variables in the list are separated by commas. There is no comma following the word "INPUT" and there is no comma after the last variable in the list.

The INPUT statement tells the computer to type a question mark and then wait for the user to enter data.

Values entered in response to an INPUT statement that contains more than one variable will be assigned to the variables in sequence. For example:

```
LISTNH
10 INPUT A,B,C
20 PRINT 'A =' ; A
30 PRINT 'B =' ; B
40 PRINT 'C =' ; C
99 END

READY
RUNNH
?3,6,7
A = 3
B = 6
C = 7

READY
```

If too few values are entered, a new question mark will be printed and the computer will wait for the rest of the values before it proceeds:

```
LISTNH
10 INPUT A,B,C
20 PRINT 'A =' ; A
30 PRINT 'B =' ; B
40 PRINT 'C =' ; C
99 END

READY
RUNNH
?3,6
?7
A = 3
B = 6
C = 7

READY
```

If too many values are entered, the extra values will be saved and used for the next INPUT statement. When the next INPUT statement is executed, no question mark will be printed and the computer will not wait for data to be entered. It will simply assign the leftover values to the variables specified in sequence:

```
LISTNH
10 INPUT A,B,C
20 PRINT "A =" ; A
30 PRINT "B =" ; B
40 PRINT "C =" ; C
50 INPUT D,E
60 PRINT "D =" ; D
70 PRINT "E =" ; E
99 END

READY
RUNNH
?1,2,3,4,5
A = 1
B = 2
C = 3
D = 4
E = 5

READY
```

Remember these things:

- The INPUT statement causes the computer to type a question mark.
- When the question mark appears, you must enter one value for each variable in the INPUT statement. The values are entered in the same left-to-right order as the variables appear in the INPUT statement.
- Numbers may be entered in standard or floating-point notation. Type commas between values.
- After entering the last number, press the RETURN key. If you have done everything correctly, the computer will proceed.

Exercise 12. The area of a triangle is found by multiplying ½ (or 0.5) times the length of its base (B) times its height (H).

```
5 PRINT "RADIUS";
10 INPUT R
15 PRINT "RADIUS", "AREA"
20 PRINT R, 3.14*R^2
99 END
```

H

B

AREA = ½BH

Write a program that asks you to enter B and H and then prints the area of the triangle with those dimensions.

Use your program to complete the following table.

B	H	Area
7.31	6.04	______
82	127	______
5×10^4	9×10^5	______
23.491	17.260	______

THE GO TO STATEMENT

The following program appeared on page 3-12.

```
LISTNH
5 PRINT "RADIUS";
10 INPUT R
15 PRINT "RADIUS", "AREA"
20 PRINT R, 3.14*R^2
99 END
READY

RUNNH
RADIUS?2
RADIUS        AREA
 2             12.56
READY

RUNNH
RADIUS?3
RADIUS        AREA
 3             28.26
READY

RUNNH
RADIUS?8
RADIUS        AREA
 8             200.96
READY
```

When you used it, you had to type RUN for **each** value of R (see page 3-12). To eliminate the need to type RUN for each new value of R, add the following GO TO statement:

30 GO TO 5   This directs the computer to "GO TO statement 5" for the next instruction.

The program now looks like this:

```
5 PRINT "RADIUS";
10 INPUT R
15 PRINT "RADIUS", "AREA"
20 PRINT R, 3.14*R^2
30 GO TO 5
99 END
```

Here is the GO TO statement.

Here is a RUN of the modified program:

```
RUNNH
RADIUS?3.14
RADIUS        AREA
 3.14          30.9591
RADIUS?6.28
RADIUS        AREA
 6.28          123.836
RADIUS?12.56
RADIUS        AREA
 12.56         495.346
RADIUS?^C
READY
```

Each time after printing the results, the computer executes a GO TO 5 and automatically returns to the INPUT statement.

How do you tell the computer that you are finished? Hold CTRL down, press C, and release. The computer will print READY.

The GO TO statement has the general form:

line number   GO TO   line number

For example:

30   GO TO   5

line number

GO TO

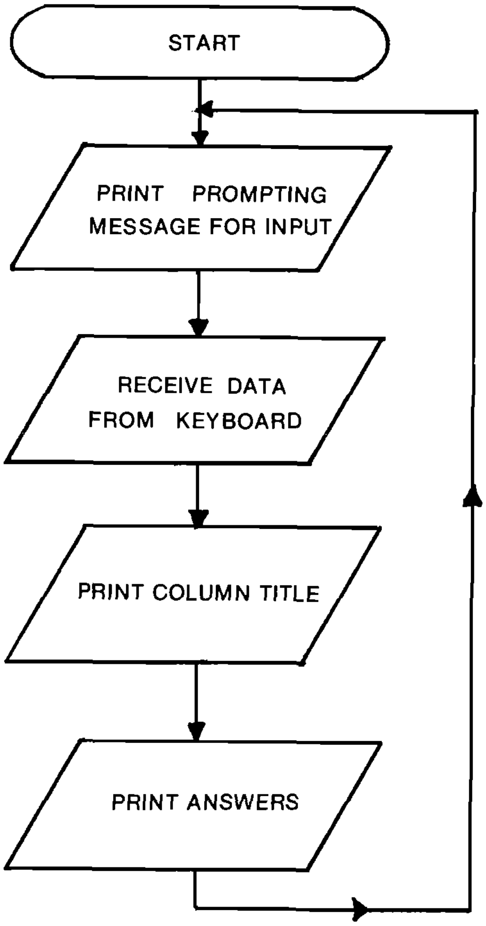
line number

The GO TO statement tells the computer to branch (transfer control) to the statement with the stated line number.

NOTE: The GO TO statement may be used either with or without the space between the O and T. That is, both of the following statements will mean the same thing to the computer:

30 GO TO 5
30 GOTO 5

The GOTO statement is best understood with the aid of a **flowchart**. A flowchart is a diagram that shows the order in which things will happen. A flowchart for the program on page 3-12 would look like this:



The symbols used in a flowchart indicate the types of processes to be executed, and the arrows show how the computer activity flows from one process to another. The trapezoid symbol (▱) is used to indicate an input or output (I/O) process. An oval (○) indicates the beginning or end of a program. A branch (GOTO) is shown by an arrow pointing to the next process to be executed. In the example, the program branches from the last statement to the first one.

Exercise 13. Complete the following program to convert from degrees Centigrade (C) to degrees Fahrenheit (F). The formula for this conversion is:

$$F = \frac{9}{5} \times C + 32$$

```
10 PRINT "CENTIGRADE TEMPERATURE";
20 INPUT C
30 LET F = [          ]
40 PRINT C; "DEG. CENT. ="; F; "DEG. FAHREN."
50 PRINT
60 GOTO [    ]
99 END
```

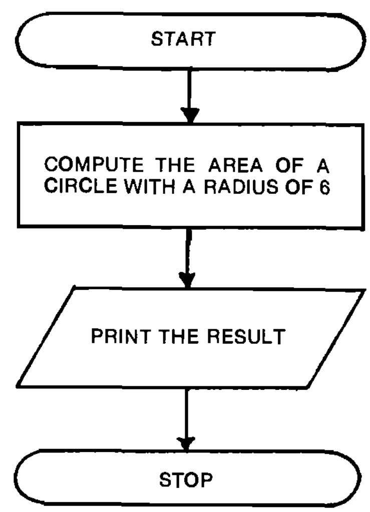
You write the formula in BASIC.

A PRINT statement without any expressions tells CLASSIC to print a blank line.
You tell CLASSIC where it should branch to.

Use your program to complete the following table:

Degrees C	Degrees F
100	______
37	______
6.8	______
0	______
-40	______
-100	______
-273.15	______

Most BASIC statements which do not involve input or output (for example, LET statements) are represented in a flowchart by rectangles. The rectangle is called the "process" symbol. For example, this flowchart:



could be translated into this program:

```
10 LET A = 3.14*6^2
20 PRINT "AREA ="; A
99 END
```

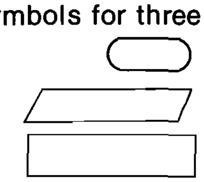
Exercise 14. Draw a flowchart of the program in Exercise 13 using the start, process, and I/O symbols.

LOOKING BACK

This section has added three more BASIC language statements to your vocabulary. You now know how to use five statements:

INPUT	GO TO
LET	END
PRINT	

You have been introduced to flowcharts and have used symbols for three different processes:



Start or Stop
Input or Output (I/O)
General Process

You also know seven editor commands:

OLD	RUN
NEW	RUNNH
SCRATCH	LIST
LISTNH	

There are many more BASIC statements to learn, but there are only three more editor commands. The next section will not teach you any new BASIC statements, but it will show you more ways to use the statements that you now know. In addition, the next section will cover two more editor commands.

SECTION 3-D

EDITING LARGER PROGRAMS

CORRECTING TYPING ERRORS

When you write larger programs, you will make more typing errors. These can be easily corrected as discussed below.

Look once again at the program that was used to find the area of a circle:

```
5 PRINT "RADIUS";
10 INPUT R
15 PRINT "RADIUS", "AREA"
20 PRINT R, 3.14*R^2
30 GOTO 5
99 END
```

You can make CLASSIC more conversational by adding more messages to this program as follows:

```
110 PRINT "THIS PROGRAM WILL FIND THE AREA OF A
120 PRINT "CIRCLE FOR WHICH THE RADIUS IS ENTER
130 PRINT
140 PRINT "ENTER BELOW THE RADIUS OF A CIRCLE:"
150 PRINT
160 PRINT "YOUR FIRST CIRCLE'S RADIUS";
170 INPUT R
180 PRINT
190 PRINT "RADIUS", "AREA"
200 PRINT R, 3.14*R^2
210 PRINT
220 PRINT "YOUR NEXT CIRCLE'S RADIUS";
230 GOTO 170
240 END
```

If you try to enter (type) this program into your workspace, it is very likely that you will make at least one typing error. If you make a typing error and notice it before you press the RETURN key, you can correct the error in two ways.

First, you can press the DELETE key.

Each press of the DELETE key causes a single character to be erased from the computer's memory, starting with the last character you typed.

When working with the editor, a short line is displayed each time you press DELETE. When you have erased back to the incorrect character, you can resume typing. For example, if you typed "DIRCLE" instead of "CIRCLE", you could correct it like this:

140 PRINT "ENTER BELOW THE RADIUS OF A DIRCLE______CIRCLE:"

A short line is displayed each time the DELETE key is pressed.

When you press the RETURN key, CLASSIC will read this line as:

140 PRINT "ENTER BELOW THE RADIUS OF A CIRCLE:"

Remember that the space is a character just like a letter or number, so it must also be deleted if typed incorrectly. For example, if you typed "BELWO THE" instead of "BELOW THE", you could correct it like this:

140 PRINT "ENTER BELWO THE______OW THE RADIUS OF A CIRCLE:"

Note that the DELETE key was pressed six times to delete the characters "WO THE".

Before you press RETURN, you may also delete the entire line by typing CTRL/U. CTRL/U is typed by holding down the CTRL key and pressing the U key. The editor will respond by printing "DELETED" and ignore the line. You may then enter the correct line as shown below:

CTRL/U typed here.

20 PRINT 3.14 DELETED
*20 PRINT R, 3.14*R∧2*

If you do not notice your error until after you have pressed the RETURN key, you must completely retype the line in error.

Suppose that you enter a line with the wrong line number. For example:

179 INPUT R

This statement must be line 170 because line 230 tells the program to "GOTO 170". You can enter the correct line 170 simply by typing it, but then you will have:

170 INPUT R
179 INPUT R

To erase line 179, simply type the line number again and press RETURN:

179 [Press RETURN Key]

and the line will be deleted.

To erase a line from the workspace, type its number followed by the RETURN key.

Exercise 15. Enter the program on page 3-15 into the workspace. If you make mistakes, use the techniques discussed to correct them. Then run the program to make sure it works. (A sample run is shown in Appendix C.)

LISTING PART OF A PROGRAM

Exercise 16. Try to LIST on the screen the program that you entered into the workspace in Exercise 15:

```
LIST

PROG1   BA    3.0

110 PRINT "THIS PROGRAM WILL FIND THE AREA OF A"
120 PRINT "CIRCLE FOR WHICH THE RADIUS IS ENTERED."
130 PRINT
140 PRINT "ENTER BELOW THE RADIUS OF A CIRCLE:"
150 PRINT
160 PRINT "YOUR FIRST CIRCLE'S RADIUS";
170 INPUT R
180 PRINT
190 PRINT "RADIUS", "AREA"
200 PRINT R, 3.14*R^2
210 PRINT
220 PRINT "YOUR NEXT CIRCLE'S RADIUS";
230 GOTO 170
240 END

READY
```

Now you will notice a problem: you cannot see the entire program on your screen at one time. This can be helped partially by using the editor LISTNH command.

The editor LISTNH command tells CLASSIC to list the program in your workspace but without printing the header (LIST No Header).

But even with the elimination of the header, the entire program will not fit on your screen. You can list part of the program by entering a line number with the LIST or LISTNH command like this:

```
LISTNH 170
170 INPUT R
180 PRINT
190 PRINT "RADIUS", "AREA"
200 PRINT R, 3.14*R^2
210 PRINT
220 PRINT "YOUR NEXT CIRCLE'S RADIUS";
230 GOTO 170
240 END

READY
```

When a LIST or LISTNH command is followed by a line number, the editor lists the program in the workspace beginning with the line number specified.

Sometimes you will want to see the first part of a program but not the later parts. To do this, type LIST or LISTNH followed by the line number at which you want the listing to start as described above. When all the lines that you are interested in have been displayed, type CTRL/O by holding down the CTRL key and pressing the letter O key.

CTRL/O tells CLASSIC to stop printing.

In the following example, CTRL/O was typed while the editor was still listing line 220:

```
LISTNH 170
170 INPUT R
180 PRINT
190 PRINT "RADIUS", "AREA"
200 PRINT R, 3.14*R^2
210 PRINT
220 PRINT "YOUR
READY
```

Remember these things:

- Pressing the DELETE key erases one character at a time, starting with the last character you typed.
- An entire line may be deleted before the RETURN key is pressed by typing CTRL/U.
- After the return key is pressed, a line containing an error may be replaced simply by retyping the line.
- A line may be deleted from the workspace by typing its line number and pressing RETURN.
- The contents of the workspace may be listed without the header by entering LISTNH.
- Part of a program may be listed by entering the LIST or LISTNH command followed by the line number at which you wish to begin the listing.
- CTRL/O will halt a listing.

Exercise 17. Experiment with the LIST and LISTNH commands by displaying on your screen various parts of the program you entered in Exercise 15. Find out the maximum number of lines that the CLASSIC screen can display at once.

CHANGING THE NAME OF THE WORKSPACE

If you began this section by responding NEW PROG1 to the NEW OR OLD— query, the name:

PROG1 BA 3.0

appeared every time you typed LIST or RUN. But the circle area program could be better named, perhaps AREA, or RADIUS, or CIRCLE. To change the name of the workspace, use the NAME command as shown below:

```
NAME CIRCLE
READY
```

You can verify that the name of the workspace has been changed by listing its contents:

```
LIST

CIRCLE  BA    3.0

110 PRINT 'THIS PROGRAM WILL FIND THE AREA OF A'
120 PRINT 'CIRCLE FOR WHICH THE RADIUS IS ENTERED.'
130 PRINT
140 PRINT 'ENTER BELOW THE RADIUS OF A CIRCLE:'
150 PRINT
160 PRINT 'YOUR FIRST CIRCLE'S RADIUS';
170 INPUT R
180 PRINT
190 PRINT 'RADIUS', 'AREA'
200 PRINT R, 3.14*R^2
210 PRINT
220 PRINT 'YOUR NEXT CIRCLE'S RADIUS';
230 GOTO 170
240 END

READY
```

The editor NAME command changes the name of the workspace.

Exercise 18. Change the name of the workspace to any of the following names and verify the change by LISTing the contents of the workspace as shown above.

AREA	ROUND
RADIUS	CURVE

SAVING PROGRAMS ON DISKS

Programs entered into the workspace can be stored on a disk with the editor SAVE command. If you store programs on the disks, you will not have to retype them every time you use the computer; they can be read into the workspace with the editor OLD command as you did with the program GUESS in Chapter 1.

Programs are stored on the disks in areas called **files.** Each file contains one program. Every file has a **file name** and a **file extension.** The file name may be up to **six** characters long, and the file extension up to **two** characters long. For example,

CIRCLE BA

file name → CIRCLE
file extension → BA

The file name is usually used to identify a specific file, while the file extension is used to indicate the **type** of the file. For example, BASIC language program files usually have the extension "BA". Therefore, when you type:

NAME CIRCLE

the editor adds the extension "BA" to the name "CIRCLE". To use a different extension, you could type:

NAME CIRCLE.JH

If this program was then saved on a disk, you would have to tell the editor its extension to read it into the workspace, like this:

OLD CIRCLE.JH

You should not use any of the following extensions because they are reserved for special use by the CLASSIC software:

Do not use the extensions:

AF	**SF**	**UF**
FF	**SV**	

If you simply type the SAVE command, the computer will write a copy of the workspace on the CLASSIC System disk with its current name.

```
SAVE
READY
```

If another program already exists on the System disk with the same name as the workspace, the above command will cause the old program to be deleted before the new one is stored. You will hear the disk click when the workspace is being copied onto it.

If you wish to save a copy of the workspace on the System disk with a name that is **different** from the name of the workspace, you can specify the name that you want after the SAVE command. For example,

SAVE ROUND

This command will cause a copy of the workspace to be stored on the System disk in a file called ROUND.BA **regardless** of the current name of the workspace.

If you wish to use an extension other than "BA", you can add the desired extension to the SAVE command. For example,

SAVE ROUND.CL

will store a copy of the workspace in a file called ROUND.CL on the System disk.

If you wish to store a program on the disk inserted in drive 1 (the right-hand disk drive), you must specify both the device name RXA1 and the name of the file to be used:

SAVE RXA1:CURVE.CL

This command will cause the workspace to be stored on the disk in RXA1 in a file called CURVE.CL. If the extension is omitted, "BA" will again be added by the system regardless of the name of the workspace.

When storing programs on RXA1, the file name must **always** be entered. The command:

SAVE RXA1:

will cause the error message:

BAD FILE

to be displayed and the command will not be executed.

Exercise 19. Using the program that you entered in Exercise 15, experiment with the SAVE command by storing this program on the System disk and on a disk inserted in drive 1. Test to see whether your program has been properly stored by trying to read it back into the workspace with the editor OLD command. If the error message BAD FILE is not printed after the OLD command, your program was properly stored on the disk.

LOOKING BACK

You have now been introduced to all but one of the BASIC editor commands. These are:

LIST	display the contents of the workspace
LISTNH	display the contents of the workspace without printing the program header
NAME	rename the workspace
NEW	clear and rename the workspace (equivalent to SCRATCH followed by NAME)
OLD	read a program into the workspace
RUN	execute the program in the workspace
RUNNH	execute the program in the workspace without printing the program header
SAVE	copy the program in the workspace onto a disk
SCRATCH	erase the workspace

The BYE command will be explained in the next section.

You should also know the special key entries recognized by the editor:

CTRL/C	return to the editor from a BASIC language program or to the monitor from the BASIC editor
CTRL/O	stop printing
CTRL/U	delete the line being typed
DELETE	delete the last character typed

The BASIC editor commands are reviewed in Chapter 3 of the *CLASSIC User's Reference Guide*. That chapter provides a quick reference for all the operations that you can perform with the editor. It also explains how each editor command can be abbreviated and what is assumed by each one.

This section has introduced many new concepts. In addition to the points made on page 3-17 you should remember these things:

- Programs may be stored on disks in areas called **files,** where each file contains one program.
- Every file has a **file name** (up to six characters long) and a **file extension** (up to two characters long).
- The name of the workspace may be changed with the editor NAME command.
- The extensions AF, FF, SF, SV, and UF are reserved for special use by the CLASSIC software and should not be used for your programs.
- A copy of the workspace can be stored on a disk with the editor SAVE command.
- If another program already exists on the disks with the same name as that used in the SAVE command, it will be erased before the new copy is stored.
- The error message BAD FILE indicates that a SAVE or OLD command was not properly executed.

Section 3-E continues to discuss files and explains how to obtain a copy of your program file on the copier.

SECTION 3-E

USING DISK FILES

GOING BACK TO THE MONITOR

Each time a program is SAVEd, its name and extension are written in the disk **directory.** The directory is like a table of contents — it contains the name, extension, and size of every file on the disk. To see the directory of your disk you must first get back to the monitor.

To get back to the monitor program from the editor, type BYE and press RETURN. The monitor program will then be read into memory (erasing any program in the workspace) and will print its dot.

```
    BYE
```

You can also return to the monitor from the editor by typing CTRL/C.

Figure 3-1 summarizes the ways to go from the monitor to the editor to a BASIC language program and back again.

To display the directory of the System disk on your screen, enter the DIRECT command to the monitor:

```
.DIRECT

BASIC  .SV    9 30-AUG-75
BRTS   .SV   15 30-AUG-75
DIRECT.SV     7 30-AUG-75
BASIC  .SF    4 30-AUG-75
BASIC  .WF    1 30-AUG-75
CCL    .SV   17 30-AUG-75
FOTP   .SV    8 30-AUG-75
BCOMP  .SV   17 30-AUG-75
BASIC  .FF    4 30-AUG-75
BASIC  .UF    4 30-AUG-75
BASIC  .AF    4 30-AUG-75
RESEQ  .BA    6 30-AUG-75
PIP    .SV   11 30-AUG-75
BLOAD  .SV    7 30-AUG-75
```

(When you enter the monitor DIRECT command on your system, the output will be different from that shown above. Also, your directory will be followed by a message indicating the amount of unused space on the disk.)

The monitor DIRECT command is used to print the directory of a disk on the terminal.

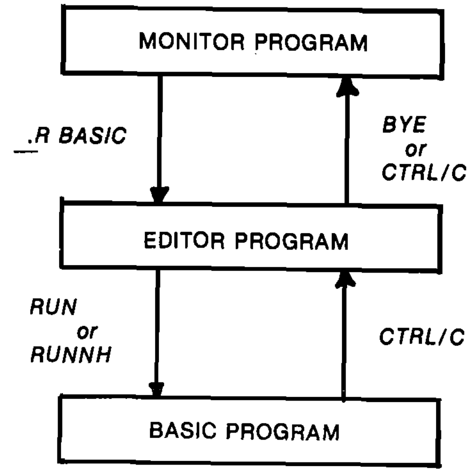


Figure 3-1

Going from the Monitor to the Editor to a BASIC Language Program and Back Again

The monitor DIRECT command in the form shown lists each file on a separate line with the file name and extension separated by a period (.). After each file name and extension, this directory shows a number and a date. The number tells the amount of space that the file occupies on the disk in units called **blocks.** If you think of a disk as a book and the directory as a table of contents, each file would be a separate chapter and each block would be a separate page. Any file takes up at least one block. The date indicates the date that the file was stored.

If you list the directory of your System disk, you may find that some files do not have dates after them. This is because CLASSIC had not been informed of the date on the day these files were stored.

The monitor DATE command informs CLASSIC of the current date or tells CLASSIC to print the current date on the screen.

Perform the following exercise to help you understand how to use the monitor DATE command.

Exercise 20. Start CLASSIC in the normal manner (Steps 1 to 8 in Chapter 1). If CLASSIC is already started, restart it by pressing the white START button again and typing S. (This will assure that the computer's memory is cleared.) Then follow these steps:

(1) List the directory of your System disk by entering the monitor DIRECT command as shown previously. Write down the name and extension of one of the files that does not have a date after it.

If all the files on your disk have dates after them, enter the following program into your workspace and SAVE it on your disk as shown:

```
.R BASIC
NEW OR OLD--NEW DATEST

READY
10 PRINT "DATEST"
99 END
SAVE

READY
BYE

.
```

List the directory on your disk again to make sure that this program has been saved without a date.

(2) Type:

.DATE

and press RETURN. CLASSIC will respond:

NONE

indicating that no date has yet been entered.

(3) Enter today's date in the form:

.DATE mm/dd/yy

where: mm is the number of this month (1-12)
dd is today's day (1-31)
yy is the last two digits of this year (oo-99)

For example,

.DATE 4/18/76

(4) Repeat Step (2). CLASSIC will respond with the date currently stored in its memory. For example,

.DATE
SUNDAY APRIL 18, 1976

.

(5) Type the following commands:

.R BASIC

NEW OR OLD—*filenam.ex*

(Substitute the file name and extension of the program that has no date where "filenam.ex" appears above, just as it appeared when you listed the directory. For example, "DATEST.BA")

READY	
SAVE	Copy the contents of the work-space back onto the disk.
READY	
BYE	Return to the monitor.

.

(6) List the directory once again:

.DIRECT

This time, the current date should be printed at the top of your directory and should follow your program name and size. For example,

CIRCLE.BA 1 18-APR-76

file name
size in blocks
creation date

Remember these things:

- The date is entered with the monitor DATE command in the form:

  .DATE mm/dd/yy

- If the monitor DATE command is entered with no date, the current date recorded by the system will be displayed. If no date is recorded, "NONE" will be printed.
- A date following a directory entry indicates the date that the corresponding file was created.

SHORTENING COMMAND LINES

For some commands, certain parameters can be left out and the system will assume **default** parameters.

Defaults are parameters that are not typed by you but are assumed by the system.

For example, you first used the monitor TYPE command in the following form:

.TYPE filnam.ex

If you try this command:

.TYPE TTY: <filnam.ex

your file will also be displayed on the screen.

The default output entry for the monitor TYPE command is "TTY:".

The short version of the monitor TYPE command is equivalent to the longer form because CLASSIC assumes "TTY:" as the default output entry. "TTY:" is the name CLASSIC calls the keyboard/screen. When no specific output entries are typed, you need not type the "< ".

The above paragraph spoke of "TTY:" as the name by which CLASSIC references the keyboard/screen. You have also seen the entry "RXA1:" that refers to the disk inserted in disk drive unit 1, the right-hand drive. Each CLASSIC unit that can be used to enter, display, or store a file is called a **device** and has a corresponding **device name.** The complete list of device names that CLASSIC will recognize is as follows:

RXA0: SYS: DSK:	disk inserted in drive unit o (on the left)
RXA1:	disk inserted in drive unit 1 (on the right)
TTY:	keyboard/screen

Whenever a file name is entered, CLASSIC assumes that the file is on DSK: (RXA0:) unless you specifically state RXA1:. For example, to run the GUESS program in Chapter 1 you entered:

NEW OR OLD—OLD RXA1:GUESS

because GUESS.BA was on the disk in drive unit 1. If you had entered:

NEW OR OLD—OLD GUESS

the system would have looked for GUESS on DSK: by **default.** If it was not found, the message:

BAD FILE

would have been printed.

For most commands, then, the default device is DSK: (RXA0:). Thus,

.DIRECT

prints the directory of the System disk inserted in drive unit 0. To obtain the directory of the disk inserted in drive unit 1, you must type:

.DIRECT RXA1:

The default output entry for the monitor DIRECT command is "TTY:" just as it is for the TYPE command. If no device is entered, "DSK:" is assumed by default.

DELETING FILES

Once a file is saved with the editor SAVE command, it can be erased from the disk by returning to the monitor and using the monitor DELETE command. This is done by typing:

.DELETE dev:filnam.ex

where "dev:filnam.ex" is the parameter of this command and indicates the name and extension of the file to be erased from the disk "dev:". If the device entry is omitted, the default assumed is "DSK:" (the same as "RXA0:" and "SYS:").

Exercise 22. Enter a short BASIC program into the workspace (such as that on page 3-20) and SAVE it on RXA1. Then use the monitor DELETE command to erase the file. DO NOT DELETE ANY FILES THAT YOU HAVE NOT PERSONALLY CREATED. List the directories of your disks before and after the deletions to assure that the files have been erased.

LOOKING BACK

With the help of this section, you should now be familiar with the following monitor commands:

DATE	inform CLASSIC of the current date or print the current date on the screen
DELETE	erase a file from a disk
DIRECT	display the directory of a disk
R BASIC	start up the BASIC editor
TYPE	display a file from a disk on the screen

Uses of these commands are summarized in Chapter 2 of the *CLASSIC User's Reference Guide.* Advanced monitor commands will be presented in Chapter 4 of this *Primer.*

The new concepts introduced in this section are as follows:

- Each disk has a directory that contains at least the name and size of each file on that disk.
- Some commands may accept **parameters** that indicate how the command is to be carried out.
- Command parameters usually have an **output entry** and an **input entry** separated by a left angle bracket ("<").
- If output or input parameters are left out of a command line, the system can sometimes assume **default** entries for the missing parameters.
- Each CLASSIC device is referred to by a device name followed by a colon (:), for example, "TTY:" means the keyboard/screen.

You are well on your way to becoming a CLASSIC programmer. The next few sections will help you learn how to use more BASIC language statements to write more sophisticated programs.

SECTION 3-F
LOOPS, DECISION POINTS, AND STRING VARIABLES

TEACHING THE COMPUTER TO COUNT

Look at the following program:

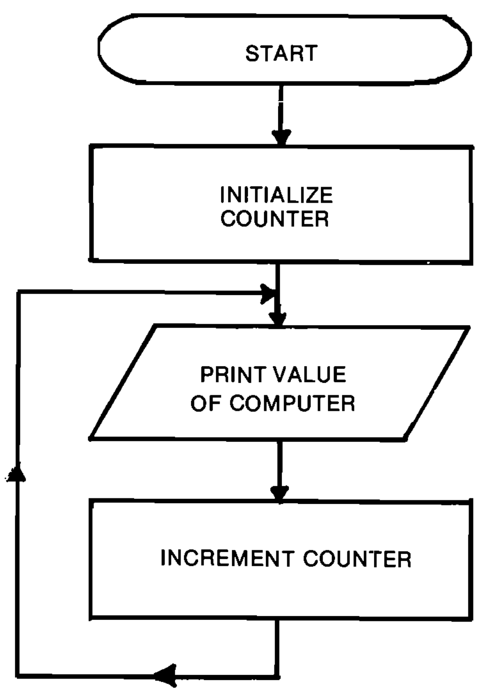
`10 LET K=1`	ASSIGN THE VALUE 1 TO K.
`20 PRINT K`	PRINT THE CURRENT VALUE OF K.
`30 LET K=K+1`	INCREASE THE VALUE OF K BY 1.
`40 GOTO 20`	GO AROUND AGAIN. IF YOU DON'T INTERRUPT THE COMPUTER, IT WILL GO ON AND ON — COUNTING NUMBERS.
`99 END`	INTERRUPT THE PROGRAM BY TYPING CTRL/C.

The above program contains a **loop:**

```
      10 LET K=1
     →20 PRINT K
LOOP  30 LET K=K+1
      40 GO TO 20
      99 END
```

THE VALUE OF K IS INITIALIZED TO 1 BEFORE THE LOOP IS EXECUTED. EACH TIME THROUGH THE LOOP, THE CURRENT VALUE OF K IS PRINTED AND INCREMENTED BY 1, AND THEN THE LOOP IS REPEATED.

This program might be translated to a flowchart like this:



The arrows show that the instructions in the last two symbols are executed over and over.

A LOOP is a set of statements that the computer executes repeatedly.

The statement:

30 LET K = K + 1

may be analyzed as follows:

Before	Statement	After
K [1]	30 LET K = K + 1	K [2]
K [2]	30 LET K = K + 1	K [3]
K [3]	30 LET K = K + 1	K [4]

Remember the general form of the LET statement:

line number LET variable = expression

The expression may be any BASIC expression. The LET statement directs the computer to evaluate the expression on the right side of the = sign and then assign the computed value to the variable on the left side of the = sign. If the expression is a variable expression, like K + 1, it is evaluated using the current value(s) of its variable(s). Therefore, the statement:

30 LET K = K + 1

directs the computer to evaluate the expression K + 1 using the current value of K and then assign the **new** value to K.

Exercise 23. Complete the following table on a separate piece of paper, showing the value that each variable will have after the statement has been executed.

Before		Statement	After
K	25	30 LET K = K + L	K [ ]
E	6	40 LET E = E + 2	E [ ]
N	4.2	200 LET N = N*5	N [ ]
X	-10	235 LET X = X + 5	X [ ]
P	0	280 LET P = P-20	P [ ]
Q	-3.1	310 LET Q = 15 + Q	Q [ ]
L	5	325 LET L = L + L + L	L [ ]
B	7	340 LET B = -B + B	B [ ]

In order to clarify what happens as the computer executes the sample program, you can "unwrap" the loop and trace it. The following table "unwraps" the loop to show the value of K following the execution of each statement in the program. Results printed by the computer are shown under the heading "OUTPUT". The program is traced seven times through the loop.

Statement	K	Output	Remarks
10 LET K=1	1		
20 PRINT K	1	1	First time through the loop.
30 LET K=K+1	2		
40 GO TO 20	2		
20 PRINT K	2	2	Second time through the loop.
30 LET K=K+1	3		
40 GO TO 20	3		
20 PRINT K	3	3	Third time through the loop.
30 LET K=K+1	4		
40 GO TO 20	4		
20 PRINT K	4	4	Fourth time through the loop.
30 LET K=K+1	5		
40 GO TO 20	5		
20 PRINT K	5	5	Fifth time through the loop.
30 LET K=K+1	6		
40 GO TO 20	6		
20 PRINT K	6	6	Sixth time through the loop.
30 LET K=K+1	7		
40 GO TO 20	7		
20 PRINT K	7	7	Seventh time through the loop.
30 LET K=K+1	8		
40 GO TO 20	8		

and so on.

Exercise 24. Trace the following program four times through the loop by filling in the blanks in the table below.

```
10 LET A=1
17 LET B=1
25 LET C=A+B
30 PRINT A
36 LET A=B
43 LET B=C
50 GOTO 25
99 END
```

Statement	A B C	Remarks
10 LET A=1	__	These statements are done once.
17 LET B=1	__ __	
25 LET C=A+B	__ __ __	First time through loop.
30 PRINT A	__ __ __	
36 LET A=B	__ __ __	
43 LET B=C	__ __ __	
50 GO TO 25		
25 LET C=A+B	__ __ __	Second time through loop.
30 PRINT A	__ __ __	
36 LET A=B	__ __ __	
43 LET B=C	__ __ __	
50 GO TO 25		

continued on next page

Statement	A B C	Remarks
25 LET C=A+B	__ __ __	Third time through loop.
30 PRINT A	__ __ __	
36 LET A=B	__ __ __	
43 LET B=C	__ __ __	
50 GO TO 25		
25 LET C=A+B	__ __ __	Fourth time through loop.
30 PRINT A	__ __ __	
36 LET A=B	__ __ __	
43 LET B=C	__ __ __	
50 GO TO 25		

Exercise 25. Without using the computer, show the first five results printed by the computer under control of each of the following programs. (Write the values that will fill in the blanks on a separate piece of paper.)

```
LISTNH
10 LET X=1
20 PRINT X
30 LET X=X+2
40 GOTO 20
99 END

READY
RUNNH
```

```
LISTNH
10 LET E=2
20 PRINT E
30 LET E=E+2
40 GOTO 20
99 END

READY
RUNNH
```

RUNNH

____
____
____
____
____

and so on.

RUNNH

____
____
____
____
____

and so on.

Exercise 26. Complete each program below (fill in the blanks on a separate piece of paper) so that when you run it, the computer will produce the results shown. Check your work with the computer.

```
LISTNH
10 LET J = [    ]
20 PRINT J
30 LET J = [      ]
40 GOTO
99 END
READY
RUNNH
 0
 1
 2
 3
 4
 5^C
READY
```

```
LISTNH
10 LET P = [    ]
20 PRINT P
30 LET P = [      ]
40 GOTO
99 END
READY
RUNNH
 1
 2
 4
 8
 16
 32
 64^C
READY
```

```
LISTNH
10 LET S = [    ]
20 PRINT S
30 LET S = [      ]
40 GOTO
99 END
READY
RUNNH
 12
 4
 1.33333
 0.444444
 0.148148^C
READY
```

SELF-STOPPING LOOPS

The loops that you have seen so far do not stop by themselves. They go on and on until you manually interrupt them by typing CTRL/C. Here is a loop that terminates automatically:

```
10 LET K=1
20 PRINT K
30 LET K=K+1
40 IF K<6 THEN 20
99 END
```

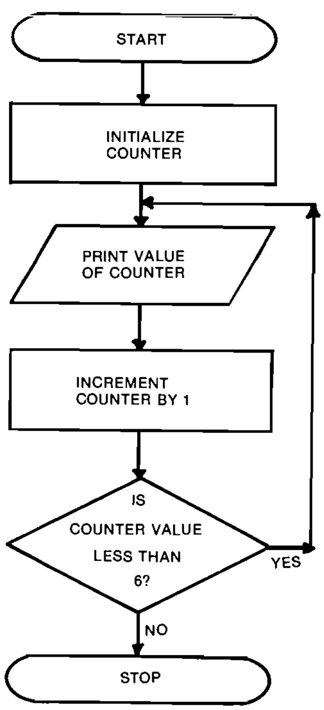
This program will print the numbers 1 to 5 and then stop:

```
RUNNH
 1
 2
 3
 4
 5

READY
```

The IF statement at line 40 causes the computer to make a decision. That is, if K **is** less than 6, the program will branch to line 20. But if K is **not** less than 6, the program will "drop through" to the statement following the IF statement.

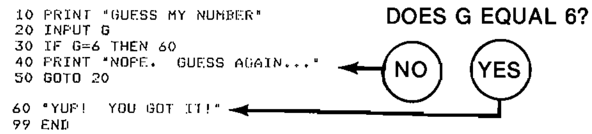
The IF statement directs the computer to examine a relation between two expressions and branch to a specified statement if and only if the relation is true. If the relation is false, the statement with the next higher line number is executed.

Decision points are represented in a flow chart by a diamond (◊). The above program would be charted as follows:



Notice that there are two paths leaving the decision symbol, one labeled "YES" and the other "NO". The path followed depends upon the truth of the relation specified in the IF statement.

Look at another example:



If G equals 6, CLASSIC executes statements 10, 20, 30, 60, and 99. If G does not equal 6, it executes 10, 20, 30, 40, and 50 and then loops back to statement 20.

The table below traces the computer's actions as it executes each statement of the program. It also shows the value of G after each statement is carried out in the following run.

```
RUNNH
GUESS MY NUMBER
?4
NOPE.   GUESS AGAIN...
?9
NOPE.   GUESS AGAIN...
?6
YUP!  YOU GOT IT!

READY
```

Statement	G	Remarks
10 PRINT "GUESS MY NUMBER?"		
20 INPUT G	4	First case: G = 4
30 IF G = 6 THEN 60	4	G = 6 is **false.**
40 PRINT "NOPE GUESS AGAIN..."	4	"Drop through" to next statement.
50 GO TO 20		Loop around.
20 INPUT G	9	Second case: G = 9.
30 IF G = 6 THEN 60	9	G = 6 is **false.**
40 PRINT "NOPE GUESS AGAIN..."	9	"Drop through" to next statement.
50 GO TO 20		Loop around.
20 INPUT G	6	Third case: G = 6.
30 IF G = 6 THEN 60	6	G = 6 is **true;** branch to statement 60.
60 PRINT "YUP! YOU GOT IT!"	6	
99 END		Program stops.

Exercise 27. Draw a flowchart for the above program.

In general, the IF statement looks like this:

n IF e_1 r e_2 THEN t

where n = line number of the IF statement

e_1 = any BASIC expression

r = any legal BASIC relation (see below)

e_2 = any BASIC expression

t = line number of the statement to be executed next if and only if the relation specified between e_1 and e_2 ("e_1 r e_2") is *true*

For example,

35	*IF*	*X*	<	*6*	*THEN*	*60*
↑	↑	↑	↑	↑	↑	↑
n	*IF*	e_1	*r*	e_2	*THEN*	*t*

X < 6 { is *true* if X is less than 6
is *false* if X is not less than 6

The following table shows the BASIC relations with their corresponding conventional relations:

Conventional	BASIC	Relation
=	=	Equal to
<	<	Less than
>	>	Greater than
≤ or ≦	< = or = <	Less than or equal to
≥ or ≧	> = or = >	Greater than or equal to
≠	< > or > <	Not equal to

NOTE: GOTO may be substituted for the word THEN in an IF statement.

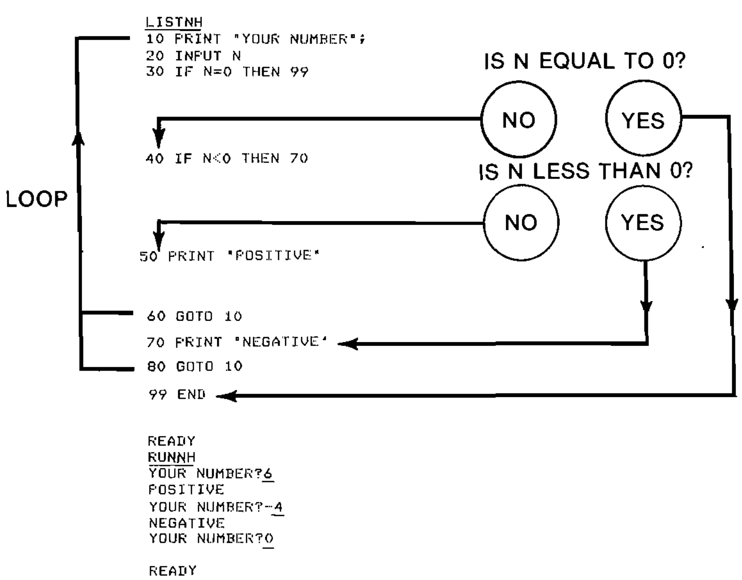
For example,

35 IF X < 6 GOTO 60

is the same as:

35 IF X < 6 THEN 60

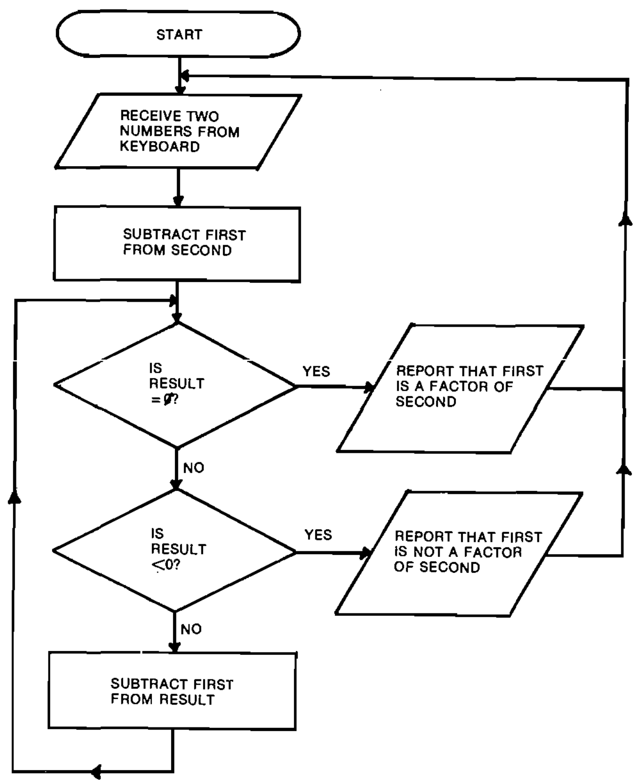
Exercise 28. The program below causes the computer to print out "positive" or "negative", depending upon the value entered. If 0 is entered, the program stops. Draw a flowchart for this program and then enter it into the computer and run it. Check that each of the paths shown in your flowchart truly reflects the actions taken by the computer.



Exercise 29. A number is said to be a **factor** of a second number if the first will divide evenly into the second without leaving a remainder. Write a program that allows you to enter two numbers and tells you whether the first is a factor of the second. Use your program to complete the following table:

First Number	Second Number	Is the First a Factor of the Second?
8	64	Yes
6	44	No
12	576	____
42	840	____
103	103	____
13	1276	____
11	6336	____
231	591	____
208	5200	____
184	1417	____
276	826	____
55	1870	____

The following flowchart will help you design your program:



STRING VARIABLES

A string variable is different from a numeric variable in two ways:

- A string variable name always ends with a dollar sign ($). For example, A$ and S$ are valid string variable names.
- A string variable may not be used in a numeric expression.

A string variable may only contain up to eight characters unless you specifically declare it to contain more (the way to do this will be discussed in Chapter 4).

The following program demonstrates a simple use of string variables:

STRING VARIABLES
STRING CONSTANTS

```
LISTNH
10 LET A$='FIRST'
20 LET B$='SECOND'
30 LET C$='THIRD'
40 PRINT A$,B$,C$
99 END

READY
RUNNH
FIRST          SECOND          THIRD

READY
```

In the above program, strings are assigned to variable locations with the LET statement. Strings may also be entered in response to an INPUT statement request:

```
LISTNH
10 PRINT 'WHAT IS YOUR NAME, PLEASE';
20 INPUT N$
30 PRINT 'HELLO, '; N$; '!'
99 END

READY
RUNNH
WHAT IS YOUR NAME, PLEASE?EVEYLN
HELLO, EVELYN!

READY
```

Each string requested by an INPUT statement must be ended by pressing the RETURN key. Therefore, if two strings are to be entered, they must be typed on separate lines. Commas, spaces, and other characters that can be used to separate numeric data **cannot** be used to separate strings. These characters will be interpreted as part of the string just like any other characters. The next example demonstrates how this works:

```
LISTNH
5 PRINT 'PLEASE ENTER YOUR FIRST AND LAST NAMES:'
7 PRINT
10 PRINT 'WHAT IS YOUR NAME, PLEASE';
20 INPUT F$,L$
30 PRINT 'HELLO, '; F$; ' '; L$; '!'
99 END

READY
RUNNH
PLEASE ENTER YOU FIRST AND LAST NAMES:

WHAT IS YOUR NAME, PLEASE?JESSE
?JAMES
HELLO, JESSE JAMES!

READY
```

If you try to assign a string longer than eight characters to a normal string variable location, the SL (String too Long) error message will be generated.

Exercise 30. Enter the above program into the workspace, but add the statement:

40 GOTO 7

Then use this program to experiment with string variables by entering strings that are of varying lengths and contain special characters on the keyboard. If you like, modify the program to experiment further.

STRING VARIABLES IN IF STATEMENTS

The expressions compared in an IF statement may contain strings as well as numbers, but evaluating the relationship between two strings can be a very involved process. The discussion here will be limited to evaluating only the equality relationship; inequalities will be discussed in Chapter 4.

Two strings are said to be equal if they contain the same characters in the same order (including blanks and punctuation).

For example, the following program will display the word "EQUAL":

```
LISTNH
10 LET A$="YES"
20 LET B$="YES"
30 IF A$=B$ THEN 60
40 PRINT "NOT EQUAL"
50 GOTO 99
60 PRINT "EQUAL"
99 END

READY
RUNNH
EQUAL

READY
```

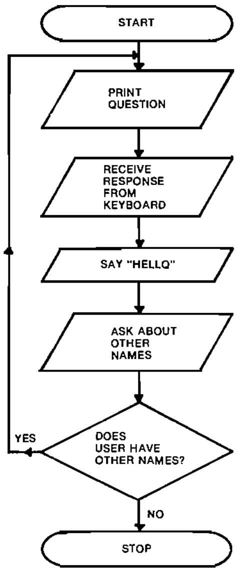
The next example will print "NOT EQUAL":

```
LISTNH
10 LET A$="YES"
20 LET B$="Y E S"
30 IF A$=B$ THEN 60
40 PRINT "NOT EQUAL"
50 GOTO 99
60 PRINT "EQUAL"
99 END

READY
RUNNH
NOT EQUAL

READY
```

In an IF statement, the contents of a string variable are usually compared to a specified string. For example,

```
LISTNH
10 PRINT 'WHAT IS YOUR NAME, PLEASE';
20 INPUT N$
30 PRINT 'HELLO, '; N$; '!'
40 PRINT 'DO YOU GO BY ANY OTHER NAMES';
50 INPUT A$
60 PRINT
70 IF A$='YES' THEN 10
99 END

READY
RUNNH
WHAT IS YOUR NAME, PLEASE?SCOTT
HELLO, SCOTT!
DO YOU GO BY ANY OTHER NAMES?YES

WHAT IS YOUR NAME, PLEASE?DOUGLAS
HELLO, DOUGLAS!
DO YOU GO DY ANY OTHER NAMES?NO

READY
```

A flowchart will help you follow this program:



Exercise 31. Modify the program that you wrote for Exercise 29 (page 3-25) to ask the user if he or she has another number to enter before it recycles for additional input. Ask the user for a simple "YES" or "NO" response and use a string variable to store that response.

LOOKING BACK

You have learned one new BASIC statement in this section, the IF statement. You have seen how this statement is used to create self-stopping loops and to evaluate relationships between both numeric and string variables. In addition, you have learned how to use variables to store strings up to eight characters long.

Remember these things:

- A loop is a set of statements that is executed repeatedly.
- The IF statement is used to examine a relation between two expressions. If the relation is true, the computer branches to a specified statement. If it is false, execution "drops through" to the next statement.
- Decision points are represented in a flow chart with a diamond-shaped symbol.
- String variable names always end with a dollar sign ($).
- A string variable may not be used in a numeric expression.
- A standard string variable may not contain more than eight characters.
- Each string entered in response to an INPUT request must be ended by pressing the RETURN key.
- Strings are equal if they are exactly the same.

The next section will show you another way to create program loops using fewer statements and a more flexible format.

SECTION 3-G

CREATING FOR-NEXT LOOPS

THE FOR AND NEXT STATEMENTS

Loops made with the IF-THEN statement require you to keep track of the number of times the loop is executed. You saw the following program on page 3-24.

```
10 LET K=1
20 PRINT K
30 LET K=K+1
40 IF K<6 THEN 20
99 END
```

This program printed the numbers 1 to 5. The following program is shorter, but will do the same thing. It uses a FOR and a NEXT statement to add 1 to K automatically each time the loop is executed:

```
LISTNH
10 FOR K=1 TO 5 }
20 PRINT K      } FOR-NEXT LOOP.
30 NEXT K       }
99 END

READY
RUNNH
 1
 2
 3
 4
 5

READY
```

Every FOR statement must have a NEXT statement and every NEXT statement must have a FOR statement.

The flowchart for this program might look exactly the same as the one on 3-24, but use of a FOR-NEXT loop eliminates one statement in the program as compared to the version using the IF statement. The following trace will help you understand how a FOR-NEXT loop works:

Statement	K	Output	Remarks
10 FOR K=1 TO 5	1		K starts at 1.
20 PRINT K	1	1	First time through loop.
30 NEXT K	2		K<5. Do it again.
20 PRINT K	2	2	Second time through loop.
30 NEXT K	3		K<5. Do it again.
20 PRINT K	3	3	Third time through loop.
30 NEXT K	4		K<5. Do it again.
20 PRINT K	4	4	Fourth time through loop.
30 NEXT K	5		K<5. Do it again.
20 PRINT K	5	5	Fifth time through loop.
30 NEXT K	6		K>5. Stop the loop and reset K to the terminal value.
99 END	5		Everything stops.

A FOR-NEXT loop consists of three things:

(1) a FOR statement,
(2) a NEXT statement, and
(3) one or more statements between the FOR statement and the NEXT statement.

A FOR-NEXT loop begins with a FOR statement and ends with a NEXT statement. The set of statements between FOR and NEXT is called the *body* of the loop.

BODY OF THE LOOP

```
10 FOR X=1 TO 12

50 NEXT X
```

THE **SAME** VARIABLE MUST BE USED AS AN INDEX IN BOTH PLACES.

HERE IS ANOTHER EXAMPLE:

```
LISTNH
10 FOR N=2 TO 7
20 PRINT N
30 NEXT N
99 END

READY
RUNNH
 2
 3
 4
 5
 6
 7

READY
```

THIS FOR STATEMENT DEFINES A **SET** OF VALUES FOR THE INDEX VARIABLE N. THE SET IS:
[2,3,4,5,6,7]

THE BODY OF THE LOOP IS EXECUTED REPEATEDLY, ONCE FOR EACH VALUE OF N DEFINED BY THE FOR STATEMENT. NOTE THAT THE INDEX IS INCREMENTED BY 1 EACH TIME THROUGH THE LOOP.

Exercise 32. Write down the numbers of the statements which make up the body of the loop in the following program:

```
LISTNH
10 PRINT "RADIUS", "AREA"
20 FOR R=2 TO 4
30 LET A=3.14*R^2
30 PRINT R,A
40 NEXT R
99 END

READY
RUNNH
RADIUS          AREA
 2               12.56
 3               28.26
 4               50.24

READY
```

Exercise 33. The volume of a sphere may be represented by the equation:

$$V = \frac{4}{3}\pi r^3$$

Write a program to display a table of volumes for spheres with radii from 1 to 30.

After a FOR/NEXT loop is completed, the index will be set equal to the value that it had the last time that the loop was executed. For example, the value output by statement 40 below will be 6:

```
LISTNH
5 LET A=0
10 FOR K=1 TO 6
20 LET A=A+K
30 NEXT K
40 PRINT K
99 END

READY
RUNNH
 6

READY
```

The following table shows the set of values defined for the index in each example of a FOR statement. Line numbers are omitted.

FOR Statement	Index	Set of Values for the Index
FOR J=0 TO 3	J	[0,1,2,3]
FOR I=1 TO 1	I	[1]
FOR A=3 TO 5	A	[3,4,5]
FOR X= -2 TO 2	X	[-2,-1,0,1,2]
FOR B=1 TO 0	B	Empty—the loop is skipped.

Exercise 34. Complete the following table on a separate piece of paper and then check your answers by writing programs to test each case.

FOR Statement	Variable	Set of Values for the Variable
FOR N=1 TO 6	N	______
FOR C=0 TO 5	C	______
FOR W=-3 TO 0	___	______
FOR E=12 TO 12	___	______
FOR T=7 TO 5	___	______
FOR X=.5 TO 2.5	___	______
FOR Y=1 TO 2.5	___	______
FOR Z=.5 TO 3	___	______

THE STEP CLAUSE

A variation of the FOR statement is shown in the following program.

```
LISTNH
10 FOR K=1 TO 9 STEP 2
20 PRINT K
30 NEXT K
99 END

READY
RUNNH
 1
 3
 5
 7
 9

READY
```

NOTE THE STEP CLAUSE.

THE STEP 2 CLAUSE CAUSES THE VALUE OF K TO INCREASE BY 2 (INSTEAD OF 1) EACH TIME. YOU CAN VERIFY THIS BY EXAMINING THE PRINTED RESULTS.

If the STEP clause is omitted, an increment value of 1 is assumed.

The following table shows the set of values defined for the variable in each FOR statement. Line numbers are omitted.

FOR Statement	Values of the Variable
FOR E=0 TO 8 STEP 2	E=0,2,4,6,8
FOR E=0 TO 9 STEP 2	E=0,2,4,6,8
FOR V=1 TO 3 STEP .5	V=1,1.5,2,2.5,3
FOR W=1 TO 7 STEP 0	W=1,1,1,...
FOR X=1 TO 3 STEP 1	X=1,2,3
FOR X=1 TO 3	X=1,2,3
FOR Y=3 TO 1 STEP -1	Y=3,2,1

Exercise 35. Complete the following table on a separate piece of paper and then check your answers by writing programs to test each case.

FOR Statement	Values of the Variable
FOR T=0 TO 6 STEP 3	T= ______
FOR N=1 TO 5 STEP 1	N= ______
FOR K=100 TO 130 STEP 10	K= ______
FOR X=0 TO 1 STEP .25	X= ______
FOR E=0 TO 0 STEP 2	E= ______
FOR B=3 TO 0 STEP -1	B= ______

Exercise 36. The surface area of a sphere may be represented by the equation:

$$S = 4 \pi r^2$$

Write a program to display a table of surface areas for spheres with radii of 10, 20, 30, ..., 100.

VARIABLE FOR STATEMENTS

By using variables instead of numerals, you can obtain variable FOR statements such as the one in the following program:

```
LISTNH
10 INPUT N
20 FOR K=1 TO N
30 PRINT K
40 NEXT K
50 GOTO 10
99 END

READY
RUNNH
?3
 1
 2
 3
?5
 1
 2
 3
 4
 5
?0
?^C
READY
```

THE VALUE FOR N IS ENTERED.
VARIABLE FOR-NEXT LOOP.
3 IS ENTERED AS THE VALUE OF N.
FOR N=3, K=1,2,3,
5 IS ENTERED AS THE VALUE OF N.
FOR N=5, K=1,2,3,4,5
0 IS ENTERED AS THE VALUE OF N.
THE FOR LOOP IS SKIPPED BECAUSE 1   N.

Each of the three numbers in a FOR statement may be replaced with a variable:

```
             50 FOR K = I TO T STEP S
Initial value ------------^  ^      ^
Terminal Value --------------+      |
Step value -------------------------+
```

Exercise 37. Run the next program on your computer and use it to complete the following table of index values corresponding to initial, final, and step values, in FOR-NEXT loops.

```
10 PRINT "INITIAL, TERMINAL, AND STEP VALUES";
20 INPUT I,T,S
30 FOR K=I TO T STEP S
40 PRINT K
50 NEXT K
60 PRINT
70 GOTO 10
99 END
```

Initial Value	Terminal Value	Step Value	Index Values
0	1	0.2	________
10	0	3	________
2	5	2	________
6	6	3	________
0.0010	0.0013	0.0001	________
8	8	-1	________
-3	-4	-0.3	________
-4	-3	*-0.3	________
926	1852	463	________
0.01	-0.01	-0.005	________

NESTED LOOPS

It is possible to write a program containing one loop inside another. This is known as **nesting** loops.

```
LISTNH
10 PRINT "I","J"
15 PRINT
20 FOR I=1 TO 2 ----------+
30 FOR J=1 TO 3 -+ INSIDE | OUTSIDE LOOP
40 PRINT I,J     | LOOP   |
50 NEXT J <------+        |
60 NEXT I <---------------+
99 END

READY
RUNNH
I             J

1             1
1             2
1             3
2             1
2             2
2             2

READY
```

The inside loop is said to be **nested** within the outside loop.

When loops are nested, the inside loop must be completely contained within the body of the outside loop.

For example, the following nesting technique is acceptable:

```
           10 FOR K=0 TO 3 <--------------------+
           20 FOR L=2 TO 6 -+                   |
              BODY OF LOOP L | LOOP L           |
BODY OF    30 NEXT L <-------+                  | LOOP K
LOOP K     40 FOR M=1 TO 17 -+                  |
              BODY OF LOOP M | LOOP M           |
           50 NEXT M <-------+                  |
           60 NEXT K <--------------------------+
```

But this is not allowed:

```
10 FOR K=-3 TO 6 -+
20 FOR L=2 TO 4 --|------------+
     BODY OF LOOP ?            |
30 NEXT K <-------+            |
40 NEXT L <--------------------+
```

because neither loop is totally contained within the body of the other.

You can use nested loops to generate tables for operations with more than one variable. For example, the following program displays a multiplication table for all combinations of integers from 0 to 9:

```
LISTNH
100 PRINT "MULTIPLICATION TABLE"
110 PRINT
120 PRINT "  ";
+---------------------+
| 130 FOR K=0 TO 9    |
| 140 PRINT " "; K;   |  SIMPLE LOOP
| 150 NEXT K          |
+---------------------+
160 PRINT
+----------------------------+
| 170 FOR L=0 TO 9           |
| 180 PRINT L;               |
| +------------------------+ |
| | 190 FOR M=0 TO 9       | |
| | 200 IF L*M>10 THEN 220 | |  NESTED LOOPS
| | 210 PRINT " ";         | |
| | 220 PRINT L*M;         | |
| | 230 NEXT M             | |
| +------------------------+ |
| 240 PRINT                  |
| 250 NEXT L                 |
+----------------------------+
999 END

READY
RUNNH
MULTIPLICATION TABLE

    0  1  2  3  4  5  6  7  8  9
 0  0  0  0  0  0  0  0  0  0  0
 1  0  1  2  3  4  5  6  7  8  9
 2  0  2  4  6  8 10 12 14 16 18
 3  0  3  6  9 12 15 18 21 24 27
 4  0  4  8 12 16 20 24 28 32 36
 5  0  5 10 15 20 25 30 35 40 45
 6  0  6 12 18 24 30 36 42 48 54
 7  0  7 14 21 28 35 42 49 56 63
 8  0  8 16 24 32 40 48 56 64 72
 9  0  9 18 27 36 45 54 63 72 81

READY
```

(Lines 200 and 210 in the above program were included to **format** the output so that all numbers appeared in straight columns.)

Exercise 38. Complete the following program by specifying "C" or "Y" as the index for each of the FOR-NEXT loops. When completed properly, this program will graph the equation in line 120.

```
100 FOR □ =-5 TO 5
110 PRINT
120 LET X=Y^2
130 FOR □ =-35 TO 36
140 IF C>X THEN 170
150 PRINT " ";
160 NEXT □
170 PRINT "*";
180 NEXT □
999 END
```

Run the completed program, changing the equation in line 120 to produce different graphs. For example, these equations will produce graphs that fit on the screen:

```
120 LET X=(Y^3)/4

120 LET X=7*Y

120 LET X=2*(Y^2)-30

120 LET X=-(Y^2)
```

LOOKING BACK

This section has introduced the technique of creating loops with the FOR and NEXT statements.

Remember these things:

- A loop made with the FOR and NEXT statements is usually at least one statement shorter than a similar loop made with the IF statement.
- The FOR statement defines a set of values for the loop index by specifying the initial, terminal, and step values of that index.
- The body of a loop is executed once for each member of the set defined by the FOR statement.
- Every FOR statement must have a corresponding NEXT statement which uses the same index variable.
- If not specified, the STEP value of a FOR-NEXT loop is assumed by the system to be +1.
- The NEXT statement causes the body of the loop to be executed again, using the next member of the set. However, if all members of the set have already been used, then the NEXT statement directs the computer to go to the statement following the NEXT statement.
- When loops are nested, the inside loop must be contained totally within the body of the outside loop.

The next section introduces another way to enter data to a program and two more ways to name variables.

SECTION 3-H

SUPPLYING LARGER AMOUNTS OF DATA

THE READ AND DATA STATEMENTS

When programs require a large amount of data, it is sometimes more convenient to use a **data table** than to supply the data through INPUT or LET statements. A data table is created by using DATA statements, and this data is entered into the program by means of READ statements. The following program is a modification of the program on page 3-13 that calculated the area of a circle, and demonstrates the use of a data table.

```
LISTNH
15 PRINT 'RADIUS', 'AREA'
17 READ R                     <---- THIS IS A READ STATEMENT.
20 PRINT R, 3.14*R^2
30 GOTO 17
40 DATA 12.2, 17.3, 29.6      <---- THIS IS A DATA STATEMENT.
99 END

READY
RUNNH
RADIUS        AREA            <---- HERE ARE THE RESULTS.
 12.2          467.357
 17.3          939.77
 29.6          2751.14

DA AT LINE 00017              <---- THIS MESSAGE MEANS
                                    THAT THE COMPUTER HAS
READY                               READ ALL THE DATA.
```

The statement:

17 READ R

tells the computer to read **one** value of R from the list of values in the DATA statement. Each time the READ statement is executed, the computer reads the **next** value from the DATA statement. In other words, the computer remembers what value should be read next.

If there is no more data to be read in the DATA statement, the computer prints the "DA" message and stops automatically.

Here is another example using the READ and DATA statements:

Four students have each taken three quizzes. Their scores are:

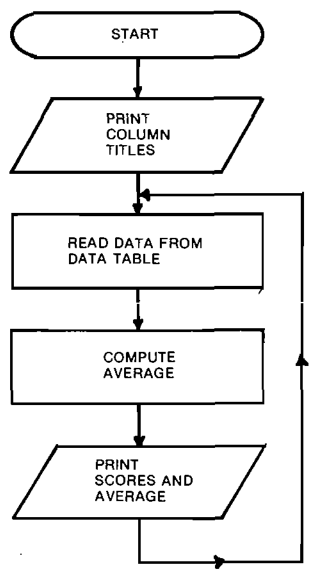
	First Score	Second Score	Third Score
First Student	66	81	75
Second Student	91	88	95
Third Student	78	78	62
Fourth Student	80	83	86

The following program computes the average of the three scores for each student:

```
LISTNH
10 PRINT 'FIRST','SECOND','THIRD'
20 PRINT 'SCORE','SCORE','SCORE','AVERAGE'
30 READ X,Y,Z
40 LET M=(X+Y+Z)/3
50 PRINT X,Y,Z,M              X, Y, AND Z DENOTE THE
60 GOTO 30                    FIRST, SECOND, AND THIRD
90 DATA 66,81,75              SCORE. M IS THE AVERAGE.
92 DATA 91,88,95
94 DATA 78,78,62              THIS IS THE DATA TABLE.
96 DATA 80,83,86
99 END

READY
RUNNH
FIRST        SECOND       THIRD
SCORE        SCORE        SCORE        AVERAGE
 66           81           75           74
 91           88           95           91.3333
 78           78           62           72.6666
 80           83           86           83

DA AT LINE 00030

READY
```

THE AVERAGES OF THE THREE SCORES ARE IN THIS COLUMN

Data statements are not executed by the computer, but simply place data in the computer's memory to be supplied when a READ statement is executed. Therefore, DATA statements may be placed anywhere within a program. If they are encountered during program execution, the computer ignores them and goes to the next executable statement. DATA statements do not appear in a flowchart, and the READ statement is represented as a process. The above program would be flowcharted as follows:



Exercise 39. Modify the above program to compute the average of any number of scores and display a table like this:

Number of scores	Average
3	89
5	74.4
.	.
.	.
.	.

Hint: Use a separate DATA statement for each set of scores and let the first number in the DATA statement indicate the number of scores in the set. Read this number and then use it as the terminal value in a FOR-NEXT loop. Sum the scores by using a statement of the form:

60 LET S=S+T

where S is the sum of the scores and T is an individual score.

Run your program using the following data:

Scores

First Student	82	88	97			
Second Student	66	78	71	82	75	
Third Student	82	86	100	91		
Fourth Student	72	82	73	82		
Fifth Student	61	73	67	80	84	79

Exercise 40. Modify the program that you wrote for Exercise 39 so that it will read all the sets of data to be averaged and then stop. Do this by adding a special data code, for example, "-99999," that signals the end of the data table. Then draw a flowchart for your final program.

String data may also be entered through DATA statements. When this is done, however, the string data must be enclosed in quotes ("). For example,

90 DATA "ONE", 1, "TWO", 2

Although numerical and string data can be included in the same DATA statement, you must make sure that the numbers are read into numerical variables and the letters or words are read into string variables. For example, the following statement could be used to read the data in statement 90 above:

130 READ A$, A, B$, B

but this statement could not:

10 READ C$, D$, C, D

The following program uses both numerical and string data:

```
LISTNH
10 PRINT "NAME", "AVERAGE"
20 PRINT
30 READ N$
40 IF N$="END-OF-DATA" THEN 99
50 READ X,Y,Z
60 PRINT N$, (X+Y+Z)/3
70 GOTO 30
80 DATA "HILLEL",66,81,75
82 DATA "JESSE",91,88,95
84 DATA "JO",80,83,86
86 DATA "STAN",78,78,62
88 DATA "END-OF-DATA"
99 END

READY
RUNNH
NAME          AVERAGE

HILLEL         74
JESSE          91.3333
JO             83
STAN           72.6666

READY
```

The general form of the READ statement is:

line number READ list of variables

For example:

10 READ X,Y$,Z

line number

READ

list of variables

THE VARIABLES ARE SEPARATED BY COMMAS.

The general form of the DATA statement is:

line number DATA list of data values

For example:

90 DATA 66,81,75, "DOUGLAS"

line number

DATA

list of data values

The READ statement directs the computer to read one value from the DATA statement for each variable in the READ statement.

If there are two or more DATA statements in a program, the values in the statement with the smallest line number are used first, then the data in the statement with the next smallest line number and so on.

All the data in a program are considered together as a single data table.

The following three sets of DATA statements are equivalent:

```
90 DATA 2,3,6,8,12,15,19,27,33,26,47,59

90 DATA 2,3,6
91 DATA 8,12,15,19,2
92 DATA 33,26,47,59

90 DATA 2,3,6,8,12,15
91 DATA 19,27,33,26,47,59
```

THE RESTORE STATEMENT

The RESTORE statement allows you to reuse DATA statements, beginning with the lowest numbered DATA statement in the program. An example of the use of the RESTORE statement is shown on the following page:

```
LISTNH
10 DATA 2,3,6
20 DATA 8,12
30 READ A,B.C,D,E
40 PRINT A;B;C;D;E
50 RESTORE
60 READ F,G
70 PRINT F;G
99 END

READY
RUNNH
  2   3   6   8   12
  2   3

READY
```

THE RESTORE STATEMENT AT LINE 50 ALLOWS THE READ STATEMENT AT LINE 60 TO OBTAIN VALUES FROM THE DATA STATEMENT AT LINE 10.

You can think of the computer as working with DATA statements by maintaining a **pointer** in the data table. Each time a value is read, the pointer is advanced to the next data value so that the computer knows which value to read next. The RESTORE statement resets the pointer to the beginning of the data table. Without the RESTORE statement in the above program, the "DA" **error message** would have occurred when CLASSIC tried to execute line 60, because all of the data in the data list would have already been read (the pointer would have been at the end of the data table).

Exercise 41. Write a program to decode messages written with numbers representing letters of the alphabet. For example,

20, 8, 9, 19, 32, 9, 19, 28, 20, 8, 5, 34, 4, 5, 3, 15, 4, 5, 4, 37, 13, 5, 19, 19, 1, 7, 5, 0

would be:

THIS IS THE DECODED MESSAGE

Note that any number over 26 represents a blank and 0 indicates the end of the message. Numbers 1 through 26 indicate the corresponding letters of the alphabet.

HINT: Receive the coded input via an INPUT statement, checking for the end of the message after each input. Use the code number to control how far a data table containing all the letters of the alphabet is searched and output the letter found. Reset the data table pointer after each search with the RESTORE statement.

ERROR MESSAGES

Error messages are very common occurrences. They can be caused by typing errors or problems in program execution. Most errors are easily corrected. When working with a BASIC language program, the computer tries to help you find your errors by printing a code indicating the type of error and the line in which it was found. A complete table of all the error messages that CLASSIC generates is given in Appendix E of the *CLASSIC User's Reference Guide.* Perhaps you have already seen some of these error messages when you ran programs previously or made mistakes in entering monitor and editor commands. Program error messages are usually of the form

XX AT LINE YY
or
XX YY

where XX is the error code, and
YY is the number of the line in which the error occured.

To understand these errors, look up their codes in the *Reference Guide* and correct the problem in your program.

Exercise 42. For each of the following statements, write the reason for its error (if any) on a separate piece of paper. If you do not think that a statement will cause an error message, try it out on your computer.

Incorrect Statement	Reason
10 READ, A,B,C	________
20 READ, XY	________
30 REED P,Q,R,S,T	________
40 READ A+B	________
50 READ I;J;K	________
60 READ AA,BB	________
70 READ ABC	________
80 READ 3.14	________
120 DATA 1/2,2/3,3/4	________
130 DATA A,B,C,D,E	________
140 DATA, 3.7,2.9	________

Error messages for monitor and editor commands are usually more informative. However, detailed explanations of these messages are also given in Appendix E of the *Reference Guide* to help you understand them. Each message in Appendix E is followed by a **solution code** referring to an entry in the table in Appendix F. This solution code table suggests actions that can be taken to correct the error.

OTHER VARIABLE NAMES

So far, you have used only the letters of the alphabet to name variables. Thus, you have been limited to 26 numeric variables (A through Z) and 26 alphanumeric variables (A$ through Z$). These have probably been enough names for you to use, but perhaps you had to use letters to stand for values that didn't quite match. For example, if you used S for "score", you couldn't use it for "sum" and "student number" in the same program.

CLASSIC allows you to combine a single digit with a letter to name a variable. For example,

S0   S1   S2   N1$ N9$

Thus you can now name up to 286 numeric (A-Z and A0-Z9) and 286 alphanumeric (A$-Z$ and A0$-Z9$) variables in a single program.

When a letter and a digit are combined in a variable name, the letter must precede the digit.

The following are **not** valid variable names:

CC   3G$ PX$ $A0
2J   77   42$ L$5

A modification of the averaging program on the preceding page is shown below. This version demonstrates the use of combination variable names. N1$ is the student's first name, N2$ his or her last name, S1, S2, and S3 the three scores, and M the arithmetic mean or average.

```
LISTNH
10 FOR K=1 TO 4
20 READ N1$,N2$,S1,S2,S3
30 LET M=(S1+S2+S3)/3
40 PRINT N2$; ","; N1$, M
50 NEXT K
60 DATA "DOUG","CARLSON",66,81,75
70 DATA "JANE","JONES",91,88,95
80 DATA "MARK","ROGERS",78,78,62
90 DATA "RUTH","SMITH",80,83,86
99 END

READY
RUNNH
CARLSON,DOUG      74
JONES,JANE        91.3333
ROGERS,MARK       72.6666
SMITH,RUTH        83

READY
```

SUBSCRIPTED VARIABLES

A third way to name variables is by using **subscripts.** A subscript in conventional form looks like this:

X_3

This is a subscript.

The symbol "X_3" is read "X sub 3."

In BASIC, subscripted variables are written in a slightly different way. Here is a BASIC subscripted variable:

X(3)

This is a subscript

"X(3)" is read "X sub 3".

The subscripted variables X(1), X(2), X(3), etc., each correspond to a location in the computer's memory just like simple variables:

X(1) ☐ X(2) ☐ X(3) ☐

Subscripted variables have two advantages over simple and combination variables. First, you may use subscripts outside the range of 0 to 9, for example, X(34), by using the DIM statement (*this will be explained toward the end of this section*). Second, you may refer to a subscripted variable using a variable as a subscript:

Subscripted variable

X(K)

variable subscript

If K=1, X(K) is X(1).
if K=2, X(K) is X(2).
If K=3, X(K) is X(3).

The set of data [X(1), X(2), X(3), etc.] is referred to as a **list.**

A list is made up of all the subscripted variables that have the same variable name.

Exercise 43. Using the data shown below, complete the table of values for the subscripted variables shown.

A(1)	8	B(1)	3.7	I	1
A(2)	-6	B(2)	9.2	J	2
A(3)	10	B(3)	3	K	3
A(4)	13	C(2)	4	X	4

Subscripted Variable	Value	Subscripted Variable	Value
A(1)	8	A(2*I)	____
A(I)	8	A(I+J)	____
A(K)	____	A(I+2)	____
A(X)	____	A(2*J-1)	____
B(I)	____	A(X-3)	____
B(3)	____	A(X-K+J)	____
B(J)	____	A(J*K-X)	____
C(J)	____	A(C(2))	____
B(1+I)	____	A(B(C(2)-1))	____

The following rules apply to the use of subscripts and subscripted variables:

(1) A subscript may be a number, a numeric variable, or a numeric expression.
(2) The value of a subscript must not be negative. If it is, an FM error message will be displayed and the program will stop.
(3) If the subscript is not a whole number, the computer uses the whole number part of the subscript. For example:

    X(3.7) is the same as X(3).
    If K=2.9, P(K) is P(2).

(4) The computer permits a subscript value of zero. For example, the following program will display the number 15.

```
10 LET A(0)=15
20 PRINT A(0)
30 END
```

Exercise 44. Following are two sections of a program. Write down the values that will be stored in each variable location **after** these statements have been executed.

```
10 FOR N=1 TO 4
20 LET P(N)=2 ∧ N
30 NEXT N
```

P(1) ☐ P(3) ☐
P(2) ☐ P(4) ☐

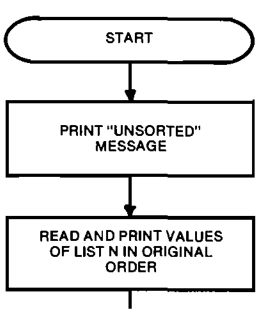
```
70 LET F(1)=1
75 FOR K=2 TO 6
80 LET F(K)=K*F(K-1)
85 NEXT K
```

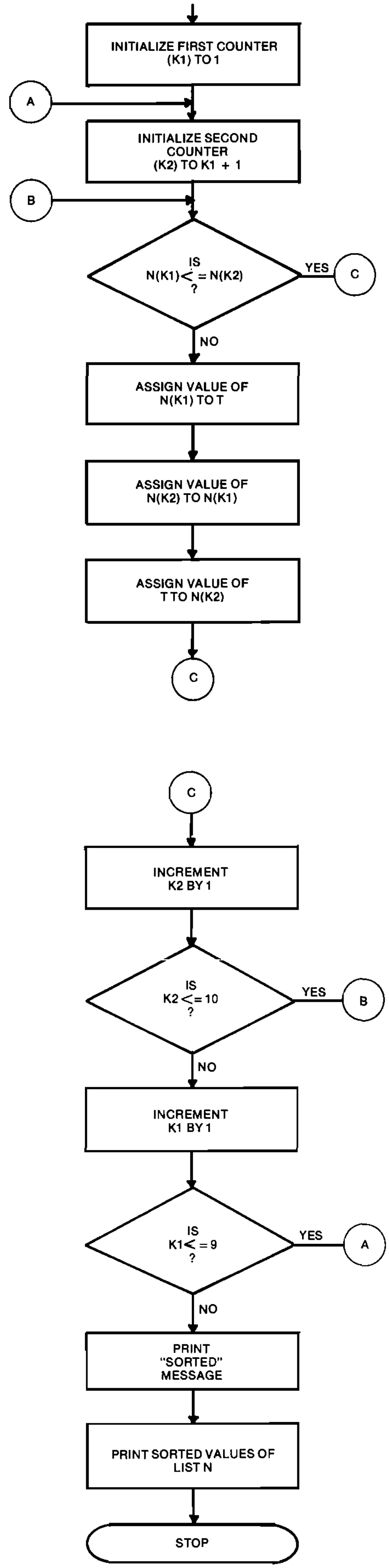
F(1) ☐ F(4) ☐
F(2) ☐ F(5) ☐
F(3) ☐ F(6) ☐

A use of subscripted variables is demonstrated by the next example. This program "sorts" numbers by placing them in order from lowest to highest. Notice the structure of the two FOR-NEXT loops at lines 170-240 and 180-230.

```
LISTNH
100 PRINT "UNSORTED DATA:"
110 FOR K=1 TO 10
120 READ N(K)
130 PRINT N(K);
140 NEXT K
150 PRINT
160 PRINT
170 FOR K1=1 TO 9
180 FOR K2=K1+1 TO 10
190 IF N(K1)<=N(K2) THEN 230
200 LET T=N(K1)
210 LET N(K1)=N(K2)
220 LET N(K2)=T
230 NEXT K2
240 NEXT K1
250 PRINT "SORTED DATA:"
260 FOR K=1 TO 10
270 PRINT N(K);
280 NEXT K
500 DATA 66,75,59,93,77,85,48,92,67,78
999 END

READY
RUNNH
UNSORTED DATA:
 66   75   59   93   77   85   48   92   67   78

SORTED DATA:
 48   59   66   67   75   77   78   85   92   93
READY
```

A flowchart for this program appears below. This flowchart uses **connectors** (letters within circles) to direct program flow to distant points.





Exercise 45. Using the program on the previous page as a model, write your own program to "invert" a list of 10 numbers. That is, given the list:

23 4 35 32 19 7 26 8 14 13

your program should output:

13 14 8 26 7 19 32 35 4 23

Use a FOR-NEXT loop and subscripted variables to perform the inversion.

LARGER DATA SETS

You may use subscripts with values from 0 to 10 for a variable in any program (A(0) through A(10)). If you wish to use values greater than 10 you must specify the largest value that your subscript will have by using the DIM (dimension) statement.

Look what happens if you try to use 11 as a subscript without dimensioning a list:

```
LISTNH
10 LET A(10)=110
20 PRINT A(10)
30 LET A(11)=111
40 PRINT A(11)
99 END

READY
RUNNH
 110

SU AT LINE 00030

READY
```

Line 30 used a subscript of 11 . . .

. . . and an error message was printed.

The following program adds the DIM statement:

```
LISTNH
5 DIM A(11)
10 LET A(10)=110
20 PRINT A(10)
30 LET A(11)=111
40 PRINT A(11)
99 END

READY
RUNNH
 110
 111

READY
```

The DIM statement at line 5 tells the computer that the subscript of A will be at most 11.

Now the program works as desired.

If you don't mention a subscripted variable in a DIM statement, the computer assumes that its subscripts will not exceed 10 in value.

A DIM statement has the following general form:

line number DIM list of subscripted variables

For example: *10 DIM A(20),B(30)*

line number

DIM

list of subscripted variables

The above DIM statement tells the computer that:

The value of any subscript of A must not exceed 20.

The value of any subscript of B must not exceed 30.

Exercise 46. Modify the program on the previous page to sort up to 100 numbers. Enter the number of numbers to be sorted as the first item in the data table and use a READ statement to assign it to a variable. Then construct numerical expressions that use this variable to specify the terminal values of the indices of the FOR-NEXT loops.

TWO-DIMENSIONAL DATA SETS

Sometimes it is convenient to organize a set of data into a two-dimensional matrix or **array.** Arrays are made by using variables with **two** subscripts. For example,

30 LET A(4,9) = 14

Each variable in an array is called an array **element.** The first subscript corresponds to the **row** number of the element and the second to its **column** number. An array dimensioned with the statement:

15 DIM A(3,4)

can be thought of as existing in the following form:

COLUMNS

ROWS

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	$A_{0,4}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,4}$

This array has 4 rows and 5 columns. Note that this is one more than the numbers of rows and columns specified in the DIM statement because the numbering begins with 0 rather than 1.

A two-dimensional array provides the easiest method for tallying survey or test data and recording how many people responded to each available option. For example, imagine that you had an eight-question survey with four possible responses to each question coded as 1, 2, 3, and 4. The response sheet for this survey might look like this:

Survey Response Sheet

Directions: Circle the code numbers corresponding to your responses for each question.

(1)	1	2	3	4	(5)	1	2	3	4
(2)	1	2	3	4	(6)	1	2	3	4
(3)	1	2	3	4	(7)	1	2	3	4
(4)	1	2	3	4	(8)	1	2	3	4

The following program tallies the number of people choosing each response for each question:

```
LISTNH
100 DIM T(8,4)
```

```
110 FOR K1=1 TO 8
120 FOR K2=1 TO 4
130 LET T(K1,K2)=0
140 NEXT K2
150 NEXT K1
```

INITIALIZING ROUTINE

```
160 READ N
```

```
170 FOR K1=1 TO N
180 FOR K2=1 TO 8
190 READ R
200 LET T(K2,R)=T(K2,R)+1
210 NEXT K2
220 NEXT K1
```

TALLYING ROUTINE

```
230 PRINT "QUESTION",,"RESPONSES"
240 PRINT ,"1","2","3","4"
```

```
250 FOR K1=1 TO 8
260 PRINT K1,
270 FOR K2=1 TO 4
280 PRINT T(K1,K2),
290 NEXT K2
300 NEXT K1
```

OUTPUT ROUTINE

```
500 DATA 5
510 DATA 2,1,4,4,1,3,1,4
520 DATA 1,1,3,4,2,4,1,3
530 DATA 2,1,4,4,1,3,1,3
540 DATA 2,2,4,4,1,3,2,1
550 DATA 4,3,4,1,3,3,2,2
999 END

READY
RUNNH
QUESTION                RESPONSES
              1         2         3         4
 1            1         3         0         1
 2            3         1         1         0
 3            0         0         1         4
 4            1         0         0         4
 5            3         1         1         0
 6            0         0         4         1
 7            3         2         0         0
 8            1         1         2         1

READY
```

The above program uses three pairs of FOR-NEXT loops. The first nested loop initializes the values of all the subscripted variables to zero. The second tallies the responses by reading a response (R) to a specific question (K2) and then using that response as the column subscript in the LET statement in line 200. The third nested loop simply outputs the response data in a format that will fit on the CLASSIC screen.

Notice that the "zero" elements (T(0,0), T(0,1), T(1,0), etc.) were not used in the above program. If you needed to conserve memory space in a very large program, you could make use of these zero elements and dimension array T as (7,3) rather than (8,4). You would then have to change the values of the indices in the loops as well. Alternatively, you might use the zero elements to store the number of people who did not respond to a specific question.

Exercise 47. Write a program to tally results on an eight-question multiple-choice test with three possible responses for each question. Output for each question the number of students who responded correctly, the number who responded incorrectly, and the number who did not respond. Use the above program as a model and store your response data in a two-dimensional array. You will need two types of data in your program, the correct responses as well as the students' responses. Enter the correct responses into a one-dimensional data list using READ and DATA statements. Use the following data to test your program (0 = no response):

Question Number:	1	2	3	4	5	6	7	8
Correct Response:	2	3	3	2	1	2	1	3

Student	Response to Questions 1	2	3	4	5	6	7	8
A	2	1	3	0	3	3	2	2
B	2	3	3	2	1	1	2	2
C	1	3	2	3	1	2	0	0
D	2	1	2	1	3	2	3	3
E	2	2	3	0	1	3	3	2
F	2	3	2	1	3	2	0	2
G	1	3	3	2	1	3	1	0
H	2	1	2	0	0	1	3	3
I	2	2	3	1	1	2	3	2
J	2	3	2	3	3	2	3	1

LOOKING BACK

You have now learned three ways to supply data to programs: through LET statements, INPUT statements, and READ/DATA statements. Each of these has advantages and disadvantages, and thus each has different applications:

- The LET statement is the most flexible because the values of expressions can be assigned to specific variable locations. However, one statement is needed for each assignment.
- The INPUT statement allows you to enter data while a program is running. It is the easiest way for other users of your program (besides yourself) to enter data because it does not require that actual BASIC language statements be changed. It is, however, relatively slow.
- The READ/DATA statements provide the fastest way for entering a large amount of data but they are relatively difficult to correct if they contain an error (the entire DATA statement must be retyped). Data statements take up room in the computer's memory and may not include any expressions that have to be evaluated.

You also know three ways to name variables: with a single letter, with a single letter followed by a single digit, and with one or two subscripts.

You can now write sizeable BASIC language programs. As your programs get larger and larger, they become more and more difficult to follow and understand, both for you and for anyone else who wishes to use your programs. The next section introduces some ways to organize your programs so that they will be easier to follow.

SECTION 3-I

ORGANIZING YOUR PROGRAMS

ADDING COMMENTS TO YOUR PROGRAMS

The sample programs that are being discussed are getting longer and longer. Flowcharts have been used to make programs easier to understand, and rectangles have been used to make the listings easier to follow. You can make your programs easier to follow by adding comments or remarks to name and separate the major sections and explain things that might confuse the reader.

The REM statement is ignored by the computer and can be used to add remarks to a program.

When the computer encounters a REM statement, it simply skips over it.

The program below is identical to the program on page 3-38, but REMarks were added to make it more readable. Note that **any** comment may be typed after the letters "REM"—the entire line is ignored by the computer.

```
LISTNH
40 REM  ***  TALLY
50 REM
60 REM     THIS PROGRAM TALLIES THE NUMBER OF PEOPLE
70 REM  CHOOSING EACH OF 4 RESPONSE FOR EACH OF 9
80 REM  QUESTIONS.
90 REM
100 DIM T(8,4)
102 REM  ARRAY 'T' STORES THE TALLY COUNTS
104 REM
106 REM  ***  INITIALIZING ROUTINE
108 REM
110 FOR K1=1 TO 8
120 FOR K2=1 TO 4
130 LET T(K1,K2)=0
140 NEXT K2
150 NEXT K1
155 REM
160 READ N
162 REM  'N' IS THE NUMBER OF SURVEYS TO BE TALLIED.
164 REM
166 REM  ***  TALLYING ROUTINE
168 REM
170 FOR K1=1 TO N
180 FOR K2=1 TO 8
190 READ R
195 REM 'R' IS A RESPONSE TO QUESTION NUMBER 'K2'.
200 LET T(K2,R)=T(K2,R)+1
210 NEXT K2
220 NEXT K1
223 REM
225 REM  ***  OUTPUT ROUTINE
227 REM
230 PRINT 'QUESTION',,'RESPONSES'
240 PRINT ,'1','2','3','4'
245 REM
250 FOR K1=1 TO 8
260 PRINT K1,
270 FOR K2=1 TO 4
280 PRINT T(K1,K2),
290 NEXT K2
300 NEXT K1
470 REM
480 REM  ***   DATA TABLE
490 REM
500 DATA 5
503 REM     THE FIRST DATA ITEM INDICATES THE NUMBER OF
505 REM  SURVEYS TO BE TALLIED.  THE ACTUAL RESPONSES
507 REM  GIVEN FOLLOW BELOW:
510 DATA 2,1,4,4,1,3,1,4
520 DATA 1,1,3,4,2,4,1,3
530 DATA 2,1,4,4,1,3,1,3
540 DATA 2,2,4,4,1,3,2,1
550 DATA 4,3,4,1,3,3,2,2
999 END

READY
```

Exercise 48. Add REMarks to the program that you wrote for Exercise 47 to make the program easier to follow and understand. Run your modified program to assure that it still works correctly.

CHANGING THE LINE NUMBERS IN YOUR PROGRAMS

As programs become modified and remarks are added, you often find that you run out of line numbers with which to add new statements. For example, it would be difficult to add another routine between lines 160 and 170 in the TALLY program at the left because most of the intervening lines numbers have already been used.

CLASSIC has a special program to **resequence** the line numbers in a program. This program is stored in the file RESEQ.BA on the system disk. To use this program, you must first store your own program on a disk with the editor SAVE command. For example,

SAVE RXA1:TALLY.BA

You can then call RESEQ into your workspace with the command:

OLD RESEQ

When you run RESEQ, the program will first ask you the name of the file you wish to resequence by printing:

FILE?

Respond to this query by entering the complete device, file name, and extension of your program (no defaults are assumed). For example,

FILE? RXA1:TALLY.BA

RESEQ will then print:

START, STEP?

and wait for you to enter the number that you wish your program to start with and the difference that you want between each successive line number (your entries must be separated by a comma).

A complete example of this procedure is shown below. (The workspace originally contained the program shown at the left.)

```
SAVE RXA1:TALLY.BA

READY
OLD RESEQ

READY
RUNNH
FILE?RXA1:TALLY.BA
START,STEP?100,10

READY
```

THE USER INDICATES THAT RESEQ SHOULD START WITH LINE NUMBER 100 AND USE AN INCREMENT OF 10 BETWEEN LINE NUMBERS.

WHEN THE READY MESSAGE REAPPEARS, YOUR PROGRAM WILL HAVE BEEN RESEQUENCED.

To read your original program back into the workspace, use the editor OLD command:

OLD RXA1:TALLY.BA

The listing on the following page shows the program at the left after it was resequenced.

When a program contains references to line numbers (such as GOTO and IF statements), the RESEQ program automatically changes these references to correspond to the new line numbers.

```
LISTNH
100 REM  ***  TALLY
110 REM
120 REM     THIS PROGRAM TALLIES THE NUMBER OF PEOPLE
130 REM  CHOOSING EACH OF 4 RESPONSE FOR EACH OF 9
140 REM  QUESTIONS.
150 REM
160 DIM T(8,4)
170 REM  ARRAY 'T' STORES THE TALLY COUNTS
180 REM
190 REM  ***  INITIALIZING ROUTINE
200 REM
210 FOR K1=1 TO 8
220 FOR K2=1 TO 4
230 LET T(K1,K2)=0
240 NEXT K2
250 NEXT K1
260 REM
270 READ N
280 REM  'N' IS THE NUMBER OF SURVEYS TO BE TALLIED.
290 REM
300 REM  ***  TALLYING ROUTINE
310 REM
320 FOR K1=1 TO N
330 FOR K2=1 TO 8
340 READ R
350 REM 'R' IS A RESPONSE TO QUESTION NUMBER 'K2'.
360 LET T(K2,R)=T(K2,R)+1
370 NEXT K2
380 NEXT K1
390 REM
400 REM  ***  OUTPUT ROUTINE
410 REM
420 PRINT 'QUESTION',,'RESPONSES'
430 PRINT ,'1','2','3','4'
440 REM
450 FOR K1=1 TO 8
460 PRINT K1,
470 FOR K2=1 TO 4
480 PRINT T(K1,K2),
490 NEXT K2
500 NEXT K1
510 REM
520 REM  ***   DATA TABLE
530 REM
540 DATA 5
550 REM     THE FIRST DATA ITEM INDICATES THE NUMBER OF
560 REM  SURVEYS TO BE TALLIED.  THE ACTUAL RESPONSES
570 REM  GIVEN FOLLOW BELOW:
580 DATA 2,1,4,4,1,3,1,4
590 DATA 1,1,3,4,2,4,1,3
600 DATA 2,1,4,4,1,3,1,3
610 DATA 2,2,4,4,1,3,2,1
620 DATA 4,3,4,1,3,3,2,2
630 END

READY
```

The RESEQ program can be used only with BASIC language programs that contain 350 or fewer lines. Any attempt to resequence a larger program will result in an error message. The RESEQ program may take as long as 10 or 15 minutes to resequence a large program, so you should not terminate it prematurely. As long as the disk drives continue to click, RESEQ is operating properly.

Exercise 49. Enter the following program into your workspace and store it on RXA1. Then use RESEQ to resequence the line numbers so that they begin with 1000 and have an increment of 10. Run the program before and after you resequence it to make sure that it works. Do not forget to SAVE this program before you OLD RESEQ, or you will have to enter it again.

```
10 READ N
30 IF N<0 THEN 90
35 IF N=0 THEN 72
50 FOR K=1 TO N
60 PRINT 'X';
70 NEXT K
71 GOTO 10
72 PRINT
73 PRINT '      ';
74 GOTO 10
90 FOR K=1 TO -N
100 PRINT ' ';
140 IF N>-20 THEN 10
150 PRINT 'PRESS RETURN:';
155 INPUT A$
200 DATA -5,6,-17,7,0,-1,4,-14,4,-5,4,0
205 DATA -1,4,-13,3,-8,3,0,-1,4,-5,5,-3,3,-8,3,0
210 DATA -1,14,-3,4,-5,4,0,15,-5,7,0
215 DATA 6,-4,5,-3,15,0,-2,3,-6,2,-6,4,-6,4,0
220 DATA -3,3,-4,2,-7,8,0,-4,3,-2,2,-8,8,0
225 DATA -5,5,-9,4,-6,4,0,-6,3,-9,15,-20
999 END
```

MULTIPLE STATEMENTS ON ONE LINE

There is one more technique that you can use to organize your programs: writing more than one BASIC statement on a single program line. To do this, you simply need to separate your statements with a backslash (\ ). This key is next to the LINE FEED key. Only the first statement in the line has a line number. For example, the following line:

*40 INPUT A \ PRINT A*12*

is equivalent to:

40 INPUT A
*50 PRINT A*12*

You can only branch to the first statement in any program line.

The message "try again" could not be printed by the following statement:

```
60 IF A$='YES' THEN 130 \ GOTO 200 \ PRINT 'TRY AGAIN'
```

because the execution of this statement will always stop before the PRINT statement is encountered. There is no way that the program can get to the PRINT statement without first hitting GOTO 200. For this reason, **GOTO statements should always be last if they are used in a multiple-statement line.**

The use of multiple statements has both advantages and disadvantages. On the plus side, the technique saves space in the computer's memory and on your disks by eliminating the need for some statement numbers. This can allow you to write larger programs. The technique also makes some statement groups (like small FOR-NEXT loops) easier to identify. On the minus side, errors are harder to correct with multiple statements on a single line because you must retype the entire line. Also, complicated formulas are often confused if several are typed on the same line. The use of this technique therefore involves some "give-and-take". It is generally a good idea to avoid multiple statements until all the "bugs" (programming errors) in your program have been found and corrected. Then go back and merge your statements to save space.

The following program demonstrates the use of multiple statements per line to shorten the TALLY program.

```
LISTNH
100 REM  ***  TALLY
110 REM
120 REM     THIS PROGRAM TALLIES THE NUMBER OF PEOPLE
130 REM  CHOOSING EACH OF 4 RESPONSE FOR EACH OF 9
140 REM  QUESTIONS.
150 REM
160 DIM T(8,4)
170 REM  ARRAY 'T' STORES THE TALLY COUNTS
180 REM
190 REM  ***  INITIALIZING ROUTINE
200 REM
210 FOR K1=1 TO 8 \ FOR K2=1 TO 4 \ LET T(K1,K2)=0
240 NEXT K2 \ NEXT K1 \ READ N
280 REM  'N' IS THE NUMBER OF SURVEYS TO BE TALLIED.
290 REM
300 REM  ***  TALLYING ROUTINE
310 REM
320 FOR K1=1 TO N \ FOR K2=1 TO 8 \ READ R
350 REM 'R' IS A RESPONSE TO QUESTION NUMBER 'K2'.
360 LET T(K2,R)=T(K2,R)+1 \ NEXT K2 \ NEXT K1
390 REM
400 REM  ***  OUTPUT ROUTINE
410 REM
420 PRINT 'QUESTION',,'RESPONSES' \ PRINT ,'1','2','3','4'
440 REM
450 FOR K1=1 TO 8 \ PRINT K1, \ FOR K2=1 TO 4
480 PRINT T(K1,K2), \ NEXT K2 \ NEXT K1
510 REM
520 REM  ***   DATA TABLE
530 REM
540 DATA 5
550 REM     THE FIRST DATA ITEM INDICATES THE NUMBER OF
560 REM  SURVEYS TO BE TALLIED.  THE ACTUAL RESPONSES
570 REM  GIVEN FOLLOW BELOW:
580 DATA 2,1,4,4,1,3,1,4
590 DATA 1,1,3,4,2,4,1,3
600 DATA 2,1,4,4,1,3,1,3
610 DATA 2,2,4,4,1,3,2,1
620 DATA 4,3,4,1,3,3,2,2
630 END

READY
```

Exercise 50. Use the multiple-statement-line technique to shorten the program given for Exercise 49 (page 3-40). Run the program after you have modified it to make sure that it still runs correctly.

SUBROUTINES

REMark statements, evenly sequenced line numbers, and multiple-statement lines all help improve a BASIC language program without changing its actual sequence or **logic.** That is, the application of any of these three techniques to a specific program would not change the flowchart describing how that program will work. A fourth technique for organizing your programs, the use of subroutines, involves arranging the actual statements in your program in a logical or **structured** manner.

A subroutine is a group of statements that might be thought of as a separate program within your main program. The use of subroutines offers three benefits in BASIC language programming:

(1) Subroutines help segment or **modularize** a program so that its general structure may be more easily followed and understood.

(2) If the same operation is performed many times within the same program, it may be easier to isolate that operation as a subroutine and branch to it whenever needed rather than repeat the same statements many times.

(3) Subroutines can be written so that they are practically little programs in themselves. For example, the sort routine discussed on page 3-36 could easily be made into a subroutine. Once this is done, this subroutine could be inserted into any program where such a sort is needed and then "called" to perform the needed operation.

The program for Exercise 49 uses two FOR-NEXT loops to perform the same operation: Printing a single character repeatedly. This program might be better structured using a subroutine:

```
900 REM  ***  MAIN PROGRAM
950 REM
1000 READ N
1010 IF N<0 THEN 1100
1020 IF N=0 THEN 1070
1023 REM
1025 REM  ***  N IS POSITIVE
1027 REM
1030 LET A$="X"
1040 GOSUB 2000
1060 GOTO 1000
1063 REM
1065 REM  ***  N IS ZERO
1067 REM
1070 PRINT
1080 PRINT '      ';
1090 GOTO 1000
1093 REM
1095 REM  ***  N IS NEGATIVE
1097 REM
1100 LET N=-N
1110 LET A$=" "
1120 GOSUB 2000
1130 IF N<20 THEN 1000
1140 PRINT 'PRESS RETURN:';
1150 INPUT A$
1152 STOP
1154 REM
1156 REM  ***  DATA TABLE
1158 REM
1160 DATA -5,6,-17,7,0,-1,4,-14,4,-5,4,0
1170 DATA -1,4,-13,3,-8,3,0,-1,4,-5,5,-3,3,-8,3,0
1180 DATA -1,14,-3,4,-5,4,0,15,-5,7,0
1190 DATA 6,-4,5,-3,15,0,-2,3,-6,2,-6,4,-6,4,0
1200 DATA -3,3,-4,2,-7,8,0,-4,3,-2,2,-8,8,0
1210 DATA -5,5,-9,4,-6,4,0,-6,3,-9,15,-20
1900 REM
1910 REM  ***  SUBROUTINE
1920 REM
2000 FOR K9=1 TO N
2010 PRINT A$;
2020 NEXT K9
2030 RETURN
9999 END
```

This program has a main routine, a data table, and a subroutine. The subroutine in lines 2000-2030 is "called" by the GOSUB statements in lines 1040 and 1120.

The GOSUB statement calls (transfers control to) a subroutine.

When a GOSUB statement calls the subroutine, the computer goes to line 2000 and executes lines 2000 to 2020. The RETURN statement in line 2030 sends the computer back to the line following the GOSUB that called the subroutine.

The RETURN statement returns control to the statement *following* the GOSUB statement that called the subroutine.

When the subroutine is called by the GOSUB at line 1040, the RETURN statement causes control to branch to statement 1060. When it is called by the GOSUB at line 1120, RETURN branches control to statement 1130.

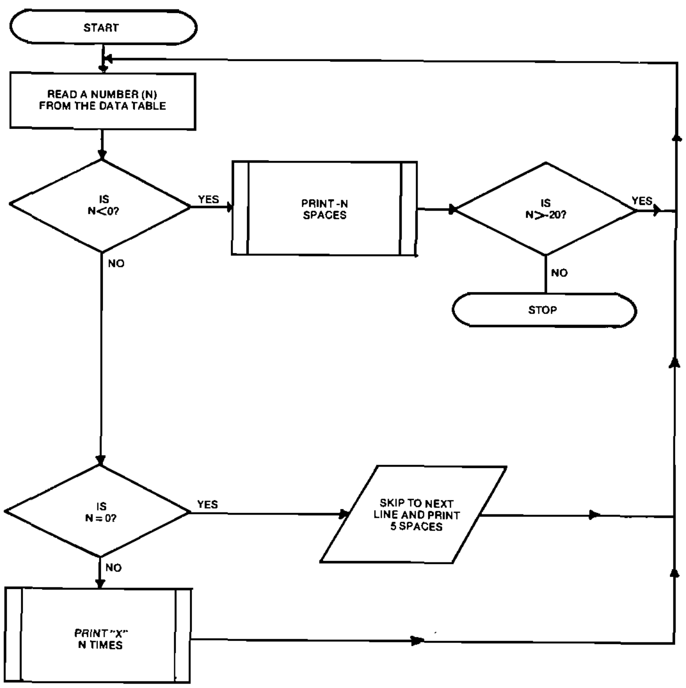
The STOP statement at line 1145 prevents the program from "falling into" the subroutine unexpectedly.

The STOP statement causes program execution to be terminated.

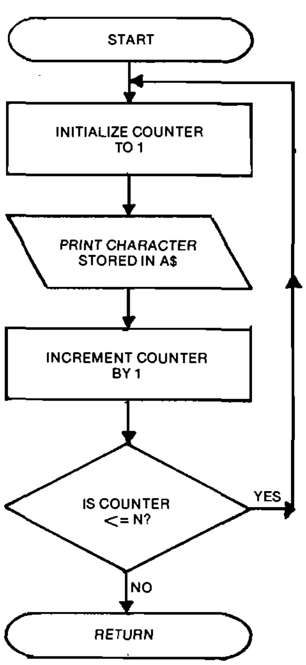
If a RETURN statement is encountered before a GOSUB is executed, a GR error message will result.

Subroutines are represented in a flowchart by a special symbol: ▯. This symbol appears in the flowchart of the main program to indicate that a subroutine is called. The actual operation of the subroutine is usually charted on a separate page. For example, flowcharts of the program, and subroutine on the previous page are shown below:

Main Program Flowchart



Subroutine Flowchart



Exercise 51. The following program was discussed on page 3-36:

```
100 PRINT "UNSORTED DATA:"
110 FOR K=1 TO 10
120 READ N(K)
130 PRINT N(K);
140 NEXT K
150 PRINT
160 PRINT
170 FOR K1=1 TO 9
180 FOR K2=K1+1 TO 10
190 IF N(K1)<=N(K2) THEN 230
200 LET T=N(K1)
210 LET N(K1)=N(K2)
220 LET N(K2)=T
230 NEXT K2
240 NEXT K1
250 PRINT "SORTED DATA:"
260 FOR K=1 TO 10
270 PRINT N(K);
280 NEXT K
500 DATA 66,75,59,93,77,85,48,92,67,78
999 END
```

Note the similarity in the loops at lines 110-140 and 260-280. Restructure this program to use a subroutine to perform the printing done by these loops.

LOOKING BACK

This chapter has introduced four techniques for organizing your programs: remarks, evenly sequenced line numbers, multiple statements on a single line, and subroutines.

Remember these things:

- Comments may be added to a program listing by means of the REM statement. These statements are ignored while the program is being executed.
- The line numbers in a program may be resequenced with the RESEQ program.
- More than one statement may be written on a single program line by separating the statements with backslashes (\ ). When this is done, however, the program can only branch to the first statement in the line.
- A subroutine can be thought of as a "program within a program" which is "called" by a GOSUB statement.
- A RETURN statement terminates a subroutine and transfers control to the statement following the last GOSUB statement that was executed.
- A STOP statement can be used to terminate the execution of a program and return control to the editor.

You now know 15 of the 25 BASIC statements available on CLASSIC. As you write your own programs, you will probably find it easier to refer to the BASIC Statement Directory in Chapter 4 of the *CLASSIC User's Reference Guide* than to refer back to this self-teaching guide. The BASIC Statement Directory presents each individual statement and lists the rules involved in using that statement. The introductory pages to that chapter review the general concepts involved in writing BASIC language programs and introduce the formats used in the directory.

This section concludes the first self-teaching chapter on the BASIC language. The next chapter is entitled

"Advanced BASIC Programming" and explains the advanced features of using BASIC on the CLASSIC system. Before you go on, make sure that you have learned all of the statements covered in this chapter by writing programs of your own which demonstrate their uses. While working the computer, use the *Reference Guide* to help you remember rules and formats that you may have forgotten.

Chapter 4
Advanced Basic Programming

SECTION 4-A
NUMERIC FUNCTIONS

A function is a special kind of subroutine. It is similar to a subroutine because it causes the computer to perform a special process. It is different from a subroutine in two ways:

(1) it is called by indicating the **function name** within a numerical or string expression (it does not require a GOSUB statement), and
(2) it requires one or more **arguments.**

Arguments are numeric values or strings that are operated on by a function.

Consider the following program. This program uses a subroutine to find the absolute value* of a number X:

```
10 INPUT X
20 GOSUB 100
30 PRINT X
40 GOTO 10
100 IF X>=0 THEN 120
110 LET X=-X
120 RETURN
999 END

READY
RUNNH
?4
 4
?-4
 4
?^C
READY
```

* For positive numbers and zero, the absolute value of a number is the same as that number. For negative numbers, the absolute value of a number is -1 times that number. For example, the absolute value of +43 is +43, but the absolute value of -18 is +18.

Note: CLASSIC does not require the word LET in variable assignment statements. That is, the following statement would be equivalent to statement number 110 above:

110 X=-X

The programs in this *Guide* use the word LET to be consistent with other BASIC language systems (see Section 5-D).

The following program finds the absolute value of X using a numeric function:

```
10 INPUT X
30 PRINT ABS(X)<---- THIS STATEMENT PRINTS
40 GOTO 10           THE ABSOLUTE VALUE OF X.
99 END

READY
RUNNH
?4
 4
?-4
 4
?^C
READY
```

The absolute value function is called by the expression:

```
             ABS(X)
function name-┘ │
argument ───────┘
```

The argument can be any number, numeric variable, or numeric expression, even one containing another function. This function is called a numeric function because its argument is numeric and it "returns" a numeric value.

Numeric functions may be used anywhere that a numeric expression is allowed.

SIMPLE NUMERIC FUNCTIONS

The square root (SQR) function. In mathematics, the symbol $\sqrt{}$ is used to indicate the square root operation. In BASIC, the corresponding symbol is SQR( ). The function SQR(X) returns the non-negative square root of the absolute value of X. Look at the following example:

```
10 PRINT SQR(4),SQR(25)
20 PRINT SQR(-4) ,SQR(-25)
99 END
RUNNH
 2              5
 2              5

READY
```

Perhaps you recall that $\sqrt{a}$ is used to mean the non-negative square root of *a*, and $-\sqrt{a}$ is used to mean the negative square root of *a*.

Here is a program to compute the two square roots of *a*:

```
10 INPUT A
20 PRINT SQR(A),-SQR(A)
30 GOTO 10
99 END
RUNNH
?4
 2             -2
?4096
 64            -64
?0
 0              0
?2
 1.41421       -1.41421
?^C
READY
```

IF A=0 THERE IS NO NEGATIVE SQUARE ROOT.
THESE ANSWERS ARE APPROXIMATIONS TO THE SQUARE ROOTS OF 2.

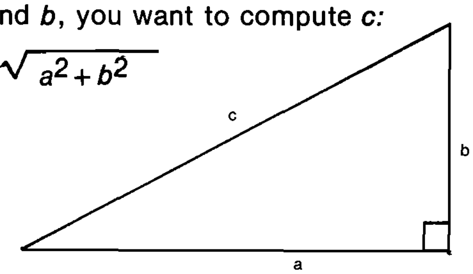
By using the FOR-NEXT loop, you can build your own square root table:

```
10 FOR X=1 TO 10
20 PRINT X,SQR(X)
30 NEXT X
99 END
RUNNH
 1              1
 2              1.41421
 3              1.73205
 4              2
 5              2.23607
 6              2.44949
 7              2.64575
 8              2.82843
 9              3
 10             3.16228

READY
```

Here is another application of the SQR function: If you know the lengths of two sides of a right triangle, you can compute the length of the third side by applying the Pythagorean theorem. For example, suppose *c* is the length of the hypotenuse and *a* and *b* are the lengths of the other two sides as indicated in the diagram below.

Given *a* and *b*, you want to compute *c:*

$$c = \sqrt{a^2 + b^2}$$



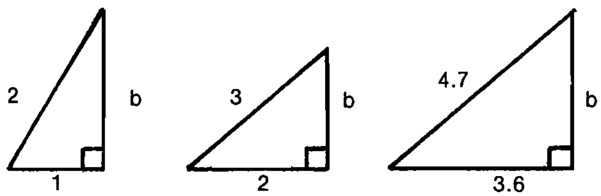

```
5 PRINT "A","B","C"
10 READ A,B
20 LET C=SQR(A^2+B^2)
30 PRINT A,B,C
40 GOTO 10
50 DATA 3,4,12,5,1,1
99 END
RUNNH
A             B             C
 3             4             5
 12            5             13
 1             1             1.41421

DA AT LINE 00010

READY
```

From the results, you see that:

If a=3 and b=4 then c=5
If a=12 and b=5 then c=13
If a=1 and b=1 then c=1.41421

Note that the argument to the SQR function at line 20 is a numeric expression. The computer evaluates the expression inside the parentheses and then executes the SQR function using the result.

Exercise 52. Suppose that you know the values of *a* and *c*. Write a program to compute and print the value of *b*. Then use your program to obtain the value of *b* for each of the following:



The sign (SGN) function. Suppose that you wanted to write a program to output the positive square root of a number when you input a positive number, and the negative square root when you input a negative number. Here is one program to do the job:

```
10 PRINT "YOUR NUMBER";
20 INPUT A
30 IF A<0 THEN 60
40 LET S=SQR(A)
50 GOTO 70
60 LET S=-SQR(A)
70 PRINT "ANSWER = ";S
80 GOTO 10
99 END

READY
RUNNH
YOUR NUMBER?4
ANSWER =  2
YOUR NUMBER?-8
ANSWER = -2.82843
YOUR NUMBER?^C
READY
```

You can rewrite this program using the SGN function, making it considerably shorter in the process.

The sign (SGN) function returns the value of -1 if the argument is negative, +1 if it is positive, and 0 if it is zero.

```
LISTNH
10 PRINT "YOUR NUMBER";
20 INPUT A
70 PRINT "ANSWER = "; SGN(A)*SQR(A)
80 GOTO 10
99 END

READY
RUNNH
YOUR NUMBER?26
ANSWER =  5.09902
YOUR NUMBER?-131
ANSWER = -11.4455
YOUR NUMBER?^C
READY
```

Exercise 53. When a number is squared, the result is always positive. Write a program to output the square of a number with the **opposite** sign that the number had originally. For example, 6 should generate -36, and -2 should generate 4.

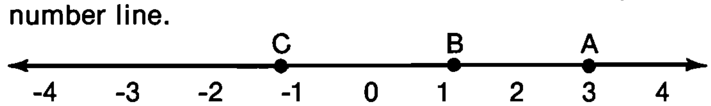
The integer (INT) function. The INT function returns the value of the largest integer **not greater than** the argument. For example,

```
10 PRINT INT(0),INT(1),INT(2),INT(3.14),INT(7.99)
99 END
READY
RUNNH
 0             1             2             3             7
READY
```

From these results, you can see that:

INT(0) = 0 INT(1) = 1 INT(2) = 2
INT(3.14) = 3 INT(7.99) = 7

The INT function is best understood with the help of a number line.



Point A is at 3. The largest integer not greater than 3 is 3. Therefore, INT(3) = 3.

Point B is at 1.25. The largest integer not greater than 1.25 is 1. Therefore, INT(1.25) = 1. Notice that if X is not an integer, the largest integer not greater than X is to the **left** of X on the number line.

Point C is at -1.25. The largest integer not greater than -1.25 is -2. Therefore, INT(-1.25) = -2. Once again, the largest integer not greater than -1.25 is to the **left** of -1.25 on the number line:

-2 < -1.25

Remember these things:

- If X is a whole number, than INT(X) = X. For example,

  INT(0) = 0 INT(1) = 1
  INT(2) = 2 INT(3) = 3
  INT(-6) = -6 INT(-4) = -4

- If X is a positive number, then INT(X) is the whole number part of X. For example,

  INT(2.99) = 2 INT(123.45)=123
  INT(0.75) = 0 INT(.05) = 0

- If X is a negative number, then INT(X) is one less than the whole number part of X. For example,

  INT(-3.6) = -4 INT(-12.4) = -13
  INT(-.3) = -1 INT(-8.8) = -9

Exercise 54. The INT function can be used to round numbers. Enter the following program into the computer and use it to round several values to the nearest whole number:

```
10 INPUT X
20 PRINT INT(X+0.5)
30 GOTO 10
99 END
```

Exercise 55. Modify the program in Exercise 54 to round numbers to the nearest tenth and the nearest hundredth. Finally, try to round a number to the nearest ten (10, 20, 30, etc.) and the nearest hundred. To do this, you will have to use a numerical expression as the function's argument and then perform a multiplication or division on the value returned.

Exercise 56. Let **x** be a 2-digit whole number. That is, **x** is a whole number such that:

$10 \leq x \leq 99$

Define a number y as follows:

y = sum of the digits of x

For example, if $x = 10$ then $y = 1 + 0 = 1$
if $x = 25$, then $y = 2 + 5 = 7$
if $x = 99$, then $y = 9 + 9 = 18$

Complete the following program to compute y for a given value of x. RUN it for the DATA shown.

```
10 READ X

20 LET Y=□

30 PRINT X,Y
40 GOTO 10
90 DATA 10,15,23,37,40,99
99 END
READY
```

Exercise 57. Let **z** be the number obtained by reversing the digits of **x**. For example:

if $x = 10$ then $z = 01 = 1$
if $x = 37$ then $z = 73$
if $x = 99$ then $z = 99$

Modify the program that you wrote for the above exercise so that the computer computes and prints the value of z instead of the value of y.

The next two parts of this section discuss logarithmic and trigonometric functions, respectively. If you have not yet studied logarithms or trigonometry, skip to the random number function on page 4-6.

LOGARITHMIC FUNCTIONS

The logarithm (LOG) function. CLASSIC computes logarithms to the base e, where e = 2.71828. These logarithms are usually referred to as **natural** logarithms. To display the natural logarithm of 6, you could use the statement:

70 PRINT LOG(6)

The LOG function computes the natural logarithm of the argument.

Very often, students begin studying logs with the base 10 rather than the base **e**. Perhaps you have seen the rule:

$$\log_a b = \frac{\log_c b}{\log_c a}$$

Since CLASSIC computes logs to the base **e**, you can find the log of N to the base 10 by substituting specific values into the above equation as follows:

$$\log_{10} N = \frac{\log_e N}{\log_e 10}$$

In BASIC, this equation would be:

10 LET L = LOG (N)/LOG(10)

where L is the log of N to the base 10.

The program below demonstrates the use of the LOG function to create a table relating natural and base 10 logs. The function itself is called at line 70.

```
10 PRINT
20 PRINT "N", "NATURAL LOG OF N", "LOG OF N TO THE BASE 10"
30 PRINT
40 FOR N=1 TO 9  STEP 1 \ GOSUB 70 \ NEXT N
50 FOR N=10 TO 90 STEP 10 \ GOSUB 70 \ NEXT N
60 LET N=100 \ GOSUB 70 \ STOP
70 PRINT N, LOG(N),, LOG(N)/LOG(10)
80 RETURN
99 END

READY
RUNNH

N              NATURAL LOG OF N          LOG OF N TO THE BASE 10

1              0                         0
2              0.693147                  0.30103
3              1.09861                   0.477121
4              1.38629                   0.60206
5              1.60944                   0.69897
6              1.79176                   0.778151
7              1.94591                   0.845098
8              2.07944                   0.90309
9              2.19722                   0.954242
10             2.30258                   1
20             2.99573                   1.30103
30             3.4012                    1.47712
40             3.68888                   1.60206
50             3.91202                   1.69897
60             4.09434                   1.77815
70             4.24849                   1.8451
80             4.38203                   1.90309
90             4.49981                   1.95424
100            4.60517                   2

READY
```

Exercise 58. Chemists measure the acidity of solutions in units called pH (potential of Hydrogen). The pH of a solution is equal to -1 times the log to the base 10 of the hydrogen ion concentration:

$$pH = -\log_{10} \text{(hydrogen ion concentration)}$$

Write a program which computes the pH of a solution when you enter a concentration.

The exponent (EXP) function. The exponent function performs exactly the opposite of the operation performed by the logarithm function. That is, given the argument N and using 2.71828 as e, the LOG function finds X in the following equation:

$$e^X = N \qquad [X = LOG(N)]$$

and the EXP function finds Y in this equation:

$$Y = e^N \qquad [Y = EXP(N)]$$

The EXP function can thus be used to convert logarithms back into regular numbers. This is called taking the **antilog** of a number. You supply a number and the EXP function will return the number whose natural logarithm equals that number.

The program below demonstrates the use of the EXP function to raise e to the Nth power and the use of the LOG function to reverse the operation of the EXP function.

```
100 PRINT
110 PRINT 'N', 'EXP(N)', 'LOG(EXP(N))', 'LOG(N)', 'EXP(LOG(N))'
120 PRINT
130 FOR N=1 TO 9 STEP 1 \ GOSUB 180 \ NEXT N
140 FOR N=10 TO 90 STEP 10 \ GOSUB 180 \ NEXT N
150 FOR N=100 TO 900 STEP 100 \ GOSUB 180 \ NEXT N
160 LET N=1000 \ GOSUB 180
170 STOP
180 PRINT N, EXP(N), LOG(EXP(N)), LOG(N), EXP(LOG(N))
190 RETURN
200 END

READY
RUNNH

N          EXP(N)        LOG(EXP(N))  LOG(N)       EXP(LOG(N))

 1          2.71828       1            0            1
 2          7.38905       2            0.693147     2
 3          20.0855       3            1.09861      3
 4          54.5981       4            1.38629      4
 5          148.413       5            1.60944      5
 6          403.428       6            1.79176      6
 7          1096.63       7            1.94591      7
 8          2980.95       8            2.07944      8
 9          8103.06       9            2.19722      8.99999
 10         22026.4       10           2.30258      9.99999
 20         .485162E+009  20           2.99573      20
 30         .106864E+014  30           3.4012       30
 40         .235382E+018  40           3.68888      40
 50         .518458E+022  50           3.91202      49.9999
 60         .114198E+027  60           4.09434      59.9999
 70         .251537E+031  70           4.24849      69.9999
 80         .554049E+035  80           4.38203      79.9999
 90         .122035E+040  90           4.49981      89.9999
 100        .268799E+044  099.999      4.60517      99.9998
 200        .722528E+087  200          5.29832      200
 300        .194219E+131  300          5.70378      299.999
 400        .522047E+174  400          5.99146      399.999
 500        .140335E+218  500          6.21461      499.998
 600        .377210E+261  600          6.39693      599.999
 700        .101400E+305  700          6.55108      699.998
 800        .272534E+348  800          6.68461      799.998
 900        .732614E+391  899.999      6.80239      899.998
 1000       .196938E+435  0999.99      6.90775      999.998

READY
```

Exercise 59. Modify the program that you wrote for Exercise 58 to convert from pH to concentration. You input a pH, and the computer should output the corresponding hydrogen ion concentration. (Hint: you will need a LOG(10) term in your expression because the EXP function uses the base **e** rather than 10.)

TRIGONOMETRIC FUNCTIONS

Angles supplied as arguments to CLASSIC trigonometric functions must always be expressed in **radians.** Radians are related to degress by the formula:

$$R = \pi D/180$$

where R is the angle measure in radians,
D is the angle measure in degrees, and
π is the constant 3.14159...

A 180 degree angle, then, is the same as a 3.14159 radian angle. The program in the next column converts degrees to radians for selected angles using the above formula.

CLASSIC has two functions that compute trigonometric values, the sine (SIN) function and the cosine (COS) function. To print the sine and cosine of an angle (A) in radians, you could use the statement:

160 PRINT SIN(A), COS(A)

Degree to Radian Conversion

```
10 PRINT
20 PRINT "ANGLE IN", "ANGLE IN"
30 PRINT " DEGREES","RADIANS"
40 PRINT
60 FOR K=0 TO 360 STEP 15
70 PRINT K, 3.14159*K/180
80 NEXT K
99 END

READY
RUNNH

ANGLE IN        ANGLE IN
 DEGREES        RADIANS

 0               0
 15              0.261799
 30              0.523598
 45              0.785397
 60              1.0472
 75              1.309
 90              1.57079
 105             1.83259
 120             2.09439
 135             2.35619
 150             2.61799
 165             2.87979
 180             3.14159
 195             3.40339
 210             3.66519
 225             3.92699
 240             4.18879
 255             4.45058
 270             4.71238
 285             4.97418
 300             5.23598
 315             5.49778
 330             5.75958
 345             6.02138
 360             6.28318

READY
```

To compute the tangent (T) of an angle (A), you simply have to divide the sine by the cosine:

50 LET T=SIN(A)/COS(A)

There is also one function to go the other way, the arctangent (ATN) function. The following statement will print the measure of the angle A (in radians) whose tangent is the number T:

60 PRINT ATN(T)

The program on the next page uses the SIN and COS functions to print a table of sines, cosines, and tangents for angles measuring between 0 and 4π radians. It then converts from the tangent back to the original angle by using the ATN function. Note that

Trigonometric Functions

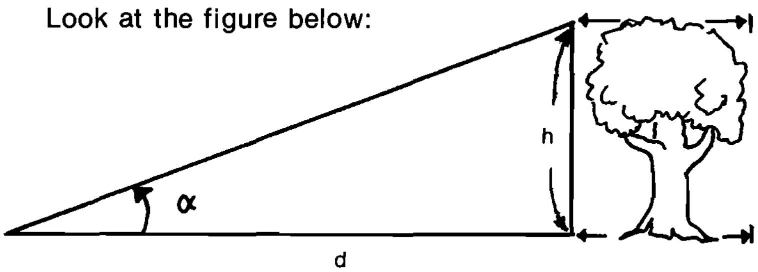
```
100 PRINT
110 PRINT "ANGLE", "SINE", "COSINE", "TANGENT", "ANGLE"
120 PRINT
130 LET P=3.14159
140 FOR K=0 TO 4*P STEP P/4
150 LET A=K
160 PRINT K, SIN(A), COS(A),
170 LET T=(SIN(A)/COS(A))
180 PRINT T, ATN(T)
190 NEXT K
200 END

READY
RUNNH

ANGLE          SINE           COSINE         TANGENT        ANGLE

 0              0              0.999999       0              0
 0.785397       0.707106       0.707108       0.999997       0.785396
 1.57079        0.999999       0.000003       333772         1.57079
 2.35619        0.70711       -0.707104      -1.00001       -0.785403
 3.14159        0.00000637    -0.999999      -0.00000637    -0.00000637
 3.92698       -0.707101      -0.707113       0.999983       0.78539
 4.71238       -0.999999      -0.00000974     102699         1.57079
 5.49777       -0.707115       0.707098      -1.00002       -0.78541
 6.28317       -0.00001273     0.999999      -0.00001273    -0.00001273
 7.06857        0.707096       0.707119       0.999969       0.785382
 7.85397        0.999999       0.00001798     55628.7        1.57078
 8.63936        0.70712       -0.707093      -1.00004       -0.785417
 9.42476        0.00001947    -0.999999      -0.00001947    -0.00001947
 10.2101       -0.707092      -0.707123       0.999957       0.785376
 10.9955       -0.999999      -0.00002397     41721.5        1.57077
 11.7809       -0.707124       0.707088      -1.00005       -0.785424
 12.5663       -0.00002547     0.999999      -0.00002547    -0.00002547

READY
```

the argument to the SIN and COS functions may be any value, but the radian angle returned by the ATN function is always in the range $-\pi/2$ to $+\pi/2$. From your math class, you may remember that an angle of $\pi/2$ (or 1.57079) radians is the same as an angle of $5\pi/2$ (or 7.85397) radians. Once again, note the limitations in CLASSIC's accuracy by comparing the values computed for these two angles in the program output.

Exercise 60. Change lines 130, 150, and 180 in the above program so that the output is generated for angles in degrees instead of radians. Use the conversion formula discussed on the previous page.

Exercise 61. Surveyors use trigonometric functions to find the heights of tall buildings and trees by a method called **triangulation.** Use the computer to perform triangulation as follows.

Look at the figure below:



By measuring the distance *d* and the angle α, one can calculate the height *h* with the formula:

$$h = d \tan \alpha$$

Write a program that allows you to input values for *d* (in meters) and α (in degrees) and outputs the corresponding value of the height h.

THE RANDOM NUMBER (RND) FUNCTION

Imagine that you flipped a coin ten times and that every time it came up "heads" you wrote "1" and every time it was "tails" you wrote "0". The numbers that you had written might look like this:

1 0 1 1 1 0 1 0 0 1

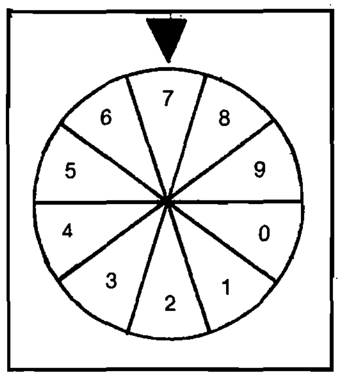
If you rolled a die and wrote down the number of spots showing on top, you might get this result:

5 2 1 5 3 6 4 2 1 4

In each case, a **random sequence** of numbers was generated. Each number in the sequence was **selected at random** from a given set of numbers. In the first case, numbers were selected at random from the set [0,1]. In the second case, they were selected from the set [1,2,3,4,5,6].

When numbers are selected at random, each number in the set has the same chance of being selected as any other member of the set. That is, the **probability** of selecting any member of the set is the same as the probability of selecting any other member.

You can obtain a random sequence of numbers from the set [0,1,2,3,4,5,6,7,8,9] by using a spinner like the one pictured below:



SPIN THE WHEEL ... SELECT THE NUMBER AT WHICH IT STOPS. THE WHEEL IS SHOWN STOPPED AT SEVEN.

Sequences of random numbers are generated by CLASSIC by using the RND function. Here is a sequence of 15 random numbers:

```
READY

10 FOR K=1 TO 15
20 PRINT RND(0),
30 NEXT K
99 END

RUNNH
.204935     .229581     .533074     .132211     .995602     .783713
.67811      .682372     .991239     .806084     .915352     .237358
.741854     .397713     .709588
READY
```

Every number in the random sequence is greater than zero but less than one. In other words,

$$0 < RND(0) < 1.$$

Every time the computer evaluates RND(0), it generates another random number between zero and one. In the above program, RND(0) occurred in a FOR-NEXT loop and was evaluated 15 times. Therefore, 15 random numbers were printed. The RND function does not require a specific argument; you may use 0 or any other number or numeric variable.

Suppose that you wanted a random sequence in which each number in the sequence is zero or one. Here is one way to get it:

```
10 FOR K=1 TO 20
20 PRINT INT(2*RND(0));
30 NEXT K
99 END
RUNNH
 0  0  1  0  1  1  1  0  1  1  0  0  1  1  0  0  1  0  1  0
READY
```

The computer prints only ones and zeroes because 2*RND(0) is always between (but never equal to) 0 and 2. That is,

*0< 2*RND(0)< 2*

The INT(2*RND(0)) can thus be only 1 or 0. Note that this statement uses one function as the argument for another (the RND function is part of the argument for the INT function).

The RND function is useful if you want to use the computer to **simulate** (imitate) a real-life activity in which chance plays a part. The following program uses random numbers to simulate flipping a coin 20 times:

```
10 FOR K=1 TO 20
20 LET R=INT(2*RND(0))
30 IF R=1 THEN 60
40 PRINT 'TAILS',
50 GOTO 70
60 PRINT 'HEADS',
70 NEXT K
99 END
```

R IS EITHER 0 OR 1. IF R=1, "HEADS" IS PRINTED. OTHERWISE, R=0 AND "TAILS" IS PRINTED.

```
READY
RUNNH
TAILS          TAILS          HEADS          TAILS          HEADS
HEADS          HEADS          TAILS          HEADS          HEADS
TAILS          TAILS          HEADS          HEADS          TAILS
TAILS          HEADS          TAILS          HEADS          TAILS

READY
```

The next program simulates dice rolling:

```
10 PRINT 'HOW MANY ROLLS';
20 INPUT T
30 PRINT
40 PRINT 'FIRST DIE','SECOND DIE','TOTAL'
50 PRINT
60 FOR K=1 TO T
65 LET A=INT(6*RND(0))+1
70 LET B=INT(6*RND(0))+1
80 PRINT A,B,A+B
90 NEXT K
99 END
RUNNH
HOW MANY ROLLS?5

FIRST DIE     SECOND DIE     TOTAL

 3             2              5
 4             3              7
 5             4              9
 6             1              7
 4             6              10

READY
```

0< 6*RND(0)< 6
THEREFORE INT(6*RND(0))
YIELDS [1,2,3,4,5,6]

Exercise 62. The **possibility set** for an expression is the complete set of values that that expression can have. The possibility set for INT(2*RND(0)) is [0,1]. What is the possibility set for each of the following expressions? (Write your answers on a separate piece of paper.)

(1) INT(3*RND(0)) ____________________
(2) INT(6*RND(0)) ____________________
(3) INT(6*RND(0)) + 1 ____________________
(4) INT(10*RND(0)) ____________________
(5) INT(10*RND(0))/10 ____________________

Look what happens when you run the program on the previous page more than once:

```
READY
RUNNH
 0.361572     0.332764     0.633057     0.350342     0.670166
 0.539795     0.8479       0.026123     0.54126      0.934326
 0.125244     0.389404     0.974853     0.516357     0.465088

READY
RUNNH
 0.361572     0.332764     0.633057     0.350342     0.670166
 0.539795     0.8479       0.026123     0.54126      0.934326
 0.125244     0.389404     0.974853     0.516357     0.465088

READY
```

The set of random numbers returned is the same both times. To get a new set of random numbers, you must use the RANDOMIZE statement.

The RANDOMIZE statement allows a new set of random numbers to be generated.

```
 2 RANDOMIZE
10 FOR K=1 TO 15
20 PRINT RND(0),
30 NEXT K
99 END

READY
RUNNH
 0.630615     0.206299     0.171631     0.126221     0.447021
 0.374268     0.0817871    0.825439     0.700928     0.354736
 0.929443     0.837158     0.392334     0.147705     0.7146

READY

READY
```

The above output is different than that in the previous column. The only difference in the program is that the RANDOMIZE statement has been added.

Remember these things:

- The RND function returns a random value between 0 and 1.
- The RANDOMIZE statement allows a new set of random numbers to be generated by the RND function.

Exercise 63. Write a program to simulate the rolling of two dice 1000 times and output the percent of times that each possible total occurs. Your output might look something like this:

```
RUNNH
TOTAL DOTS SHOWN              PERCENT OCCURRENCE
 2                             2.5
 3                             6.4
 4                             7.8
 5                             10.8
 6                             15.8
 7                             14.1
 8                             13.2
 9                             12.8
 10                            8
 11                            5.4
 12                            3.2

READY
```

Insert the RANDOMIZE statement and run the program several times to see how the results vary.

DEFINING YOUR OWN FUNCTIONS

In addition to the functions supplied by CLASSIC, you can also define your own functions. This is done by using the DEF statement. For example,

10 DEF FNA(X) =X+3

If this function has been defined, the statement:

20 PRINT FNA(6)

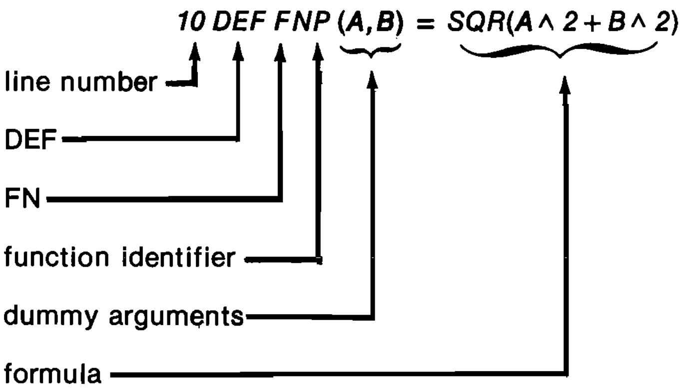
will cause the number 9 to be printed, because 6+3=9.

All user-defined functions begin with the letters FN and have one additional letter. Therefore, you can define up to 26 functions in any one program. Each function can have either one or two arguments. The variables used as the arguments in the DEF statement are called "dummy" arguments and need not have any other significance in your program; they are simply used in the definition of the operation to be carried out. The formula specified in the DEF statement may be any valid numerical expression, and it may contain up to 14 dummy arguments.

The general format of the DEF statement is as follows:

line number DEF FNa(x,y) = formula

For example,



Only the dummy arguments may be used as variables in the formula.

The next example program defines a function that converts Fahrenheit temperature to Centigrade. The math formula for this conversion is:

$$C = (F - 32) \times \frac{5}{9}$$

Note the way that this is translated into BASIC with the DEF statement:

```
10 DEF FNC(T)=(T-.32)*5/9
20 PRINT
30 PRINT "FAHRENHEIT TEMPERATURE";
40 INPUT A
50 PRINT FNC(A);"DEGREES CENTIGRADE"
60 GOTO 20
99 END
RUNNH

FAHRENHEIT TEMPERATURE?212
 117.6 DEGREES CENTIGRADE

FAHRENHEIT TEMPERATURE?32
 17.6 DEGREES CENTIGRADE

FAHRENHEIT TEMPERATURE?98.6
 54.6 DEGREES CENTIGRADE

FAHRENHEIT TEMPERATURE?^C
READY
```

The next example shows an adaption of the above program to change either Fahrenheit to Centigrade or Centigrade to Fahrenheit. This program has two DEF statements. Note the awkward way in which the words FAHRENHEIT and CENTIGRADE are assigned to locations A1\$, A2\$, B1\$, and B2\$ at lines 210 to 240 because these words are too long to fit into a single string variable (limited to 8 characters). The next section will show you a way to solve this problem more elegantly and perform other operations on strings.

```
100 REM  ***  TEMPERATURE CONVERSIONS
110 REM
120 REM  ***  FUNCTION DEFINITIONS
130 DEF FNC(T)=(T-32)*5/9
140 DEF FNF(T)=(9/5)*T+32
150 REM  ***  USER TEMPERATURE INPUT
160 PRINT \ PRINT 'TEMPERATURE';
170 INPUT TO, T$
180 IF T$='C' THEN 270
190 REM  ***  FAHRENHEIT TO CENTIGRADE
200 LET A=FNC(TO)
210 LET A1$='FAHREN'
220 LET A2$='HEIT'
230 LET B1$='CENTI'
240 LET B2$='GRADE'
250 GOTO 330
260 REM  ***  CENTIGRADE TO FAHRENHEIT
270 LET A=FNF(TO)
280 LET A1$='CENTI'
290 LET A2$='GRADE'
300 LET B1$='FAHREN'
310 LET B2$='HEIT'
320 REM  ***   ANSWER PRINT-OUT
330 PRINT TO; 'DEGREES '; A1$; A2$; ' = ';
340 PRINT A; 'DEGREES '; B1$; B2$
350 GOTO 160
360 END
READY
RUNNH

TEMPERATURE?32 F
 32 DEGREES FAHRENHEIT =  0 DEGREES CENTIGRADE

TEMPERATURE?98.6 F
 98.6 DEGREES FAHRENHEIT =  37 DEGREES CENTIGRADE

TEMPERATURE?100 C
 100 DEGREES CENTIGRADE =  212 DEGREES FAHRENHEIT

TEMPERATURE?-273.15 C
-273.15 DEGREES CENTIGRADE = -459.67 DEGREES FAHRENHEIT

TEMPERATURE?^C
READY
```

Exercise 64. When a program uses formulas that contain the same term, it is sometimes easier to define this term as a function rather than type it in several statements. For example, the following formulas all contain the term "πr":

circumference of a circle	$2\pi r = 2\cdot(\pi r)$
area of a circle	$\pi r^2 = r\cdot(\pi r)$
volume of a sphere	$\frac{4}{3} r^3 = \frac{4}{3} r^2\cdot(\pi r)$
surface area of a sphere	$4\pi r^2 = 4r\cdot(\pi r)$

Write a program that allows you to input a value for r and outputs each of the above four values. Define πr as a function and use it to evaluate this term whenever needed.

LOOKING BACK

You now know all of the numeric functions that are available in CLASSIC BASIC. They are:

- ABS returns the absolute value of an expression.
- ATN returns the angle (in radians) whose tangent is the given argument
- COS returns the cosine of the angle specified in radians
- EXP returns the value of e (2.71828) raised to the

power of the argument

FNa returns a value computed by a corresponding DEF statement

INT returns the value of the largest integer not greater than the argument

LOG returns the natural logarithm of the argument

RND returns a random number between 0 and 1

SGN returns 1 if the argument is positive, 0 if it is zero, and -1 if it is negative

SIN returns the sine of the angle specified in radians

SQR returns the square root of the argument

Before you go on to the next section, you might like to study the program below. This program plays the game of 23 Matches and uses the RND and INT functions to figure out how to beat you. Enter this program into your computer and run it. If you look at the program carefully, you might be able to figure out a strategy to beat CLASSIC.

The Game of 23 Matches

```
100 REM ***23 MATCHES
110 PRINT "LET'S PLAY 23 MATCHES. WE START WITH 23 MATCHES.
115 PRINT "YOU MOVE FIRST. YOU MAY TAKE 1,2 OR 3 MATCHES."
120 PRINT "THEN I MOVE...I MAY TAKE 1,2 OR 3 MATCHES."
125 PRINT "YOU MOVE, I MOVE AND SO ON. THE ONE WHO HAS TO"
130 PRINT "TAKE THE LAST MATCH LOSES."
135 PRINT "GOOD LUCK AND MAY THE BEST COMPUTER (HA HA) WIN."
140 PRINT
150 LET M=23
200 REM ***THE HUMAN MOVES
205 PRINT
210 PRINT "THERE ARE NOW";M;"MATCHES."
215 PRINT
220 PRINT "HOW MANY DO YOU TAKE";
230 INPUT H
240 IF H>M THEN 510
250 IF H<>INT(H) THEN 510
260 IF H<=0 THEN 510
270 IF H>=4 THEN 510
280 LET M=M-H
290 IF M=0 THEN 410
300 REM ***THE COMPUTER MOVES
305 IF M=1 THEN 440
310 LET R=M-4*INT(M/4)
320 IF R<>1 THEN 350
330 LET C=INT(3*RND(0))+1
340 GO TO 360
350 LET C=(R+3)-4*INT((R+3)/4)
360 LET M=M-C
370 IF M=0 THEN 440
375 PRINT
380 PRINT "I TOOK";C;"...";
390 GO TO 210
400 REM ***SOMEBODY WON (SEE LINES 290,305,370)
410 PRINT
420 PRINT "I WON!!! BETTER LUCK NEXT TIME."
430 GO TO 140
440 PRINT
450 PRINT "O.K. SO YOU WON. LET'S PLAY AGAIN."
460 GO TO 140
500 REM ***THE HUMAN CHEATED! (SEE LINES 240 THRU 270)
510 PRINT "YOU CHEATED! BUT I'LL GIVE YOU ANOTHER CHANCE."
520 GO TO 215
999 END

READY
```

SECTION 4-B

STRING AND SPECIAL FUNCTIONS (Part I)

LONGER STRINGS

Here is part of the temperature conversion program that you saw on the previous page:

```
280 LET A1$="CENTI"
290 LET A2$="GRADE"
300 LET B1$="FAHREN"
310 LET B2$="HEIT"
```

In this program, the words "Fahrenheit" and "Centigrade" have to be split up because they are over eight characters long. If you tried to read all the characters into a single string variable location, you would get an error message:

```
280 LET A$="CENTIGRADE"
290 LET B$="FAHRENHEIT"
330 PRINT "THIS PROGRAM CONVERTS ";A$;" TO ";B$
360 END
RUNNH

SL AT LINE 00280

READY
```

The SL error message means that a string was too long to be stored in the variable location desired.

CLASSIC normally allows a maximum of eight characters to be stored in each string variable location. By using the DIM statement, however, you can cause CLASSIC to store up to 72 characters in a single variable location:

```
145 DIM A$(10),B$(10)
280 LET A$="CENTIGRADE"
290 LET B$="FAHRENHEIT"
330 PRINT "THIS PROGRAM CONVERTS ";A$;" TO ";B$
360 END

READY


RUNNH
THIS PROGRAM CONVERTS CENTIGRADE TO FAHRENHEIT

READY
```

By adding statement 145 to the temperature conversion program and changing the string variables, the program can be simplified. In the following listing, the statements outlined by rectangles were changed from the previous version.

```
100 REM  ***  TEMPERATURE CONVERSIONS
110 REM
120 REM  ***  FUNCTION DEFINITIONS
130 DEF FNC(T)=(T-32)*5/9
140 DEF FNF(T)=(9/5)*T+32
143 REM  ***  DIMENSION OF STRING LENGTHS
145 DIM A$(10), B$(10)
148 LET A$(1)="FAHRENHEIT"
149 LET A$(2)="CENTIGRADE"
150 REM  ***  USER TEMPERATURE INPUT
160 PRINT \ PRINT "TEMPERATURE";
170 INPUT TO, T$
180 IF T$="C" THEN 270
190 REM  ***  FAHRENHEIT TO CENTIGRADE
200 LET A=FNC(TO)
210 LET A$="FAHRENHEIT"
230 LET B$="CENTIGRADE"
250 GOTO 330
260 REM  ***  CENTIGRADE TO FAHRENHEIT
270 LET A=FNF(TO)
280 LET A$="CENTIGRADE"
300 LET B$="FAHRENHEIT"
320 REM  ***  ANSWER PRINT-OUT
330 PRINT TO; "DEGREES "; A$; " = ";
340 PRINT A; "DEGREES "; B$
350 GOTO 160
360 END
```

Note that the statement:

145 DIM A$(10), B$(10)

does **not** dimension string arrays. It dimensions two strings, A$ and B$, each up to 10 characters long.

The statement DIM S$(n) dimensions a single string variable, S$, with a length of n characters.

To dimension a string list, you must supply two numbers within the parentheses of a DIM statement:

```
145 DIM A$(2,10)
148 LET A$(1)="FAHRENHEIT"
149 LET A$(2)="CENTIGRADE"
330 PRINT "THIS PROGRAM CONVERTS "; A$(1); " TO "; A$(2)
360 END

READY
RUNNH
THIS PROGRAM CONVERTS FAHRENHEIT TO CENTIGRADE

READY
```

The statement:

145 DIM A$(2,10)

does **not** set up a two-dimensional array. It dimensions 3 strings, A$(0), A$(1), and A$(2), each up to 10 characters long.

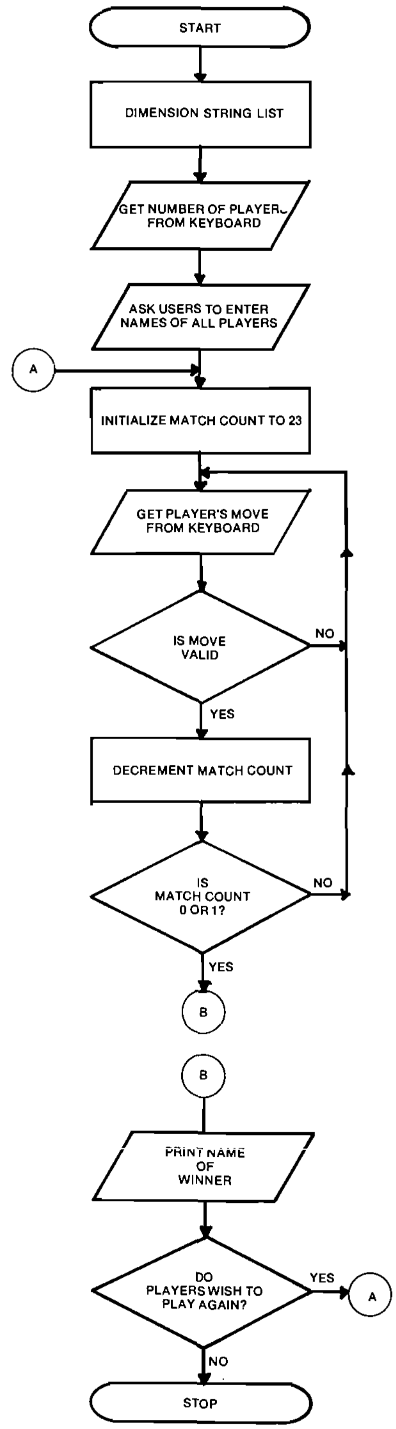
The statement DIM S$(m,n) dimensions m + 1 string variables, each up to n characters long.

The temperature conversion program can now be further modified. As before, the changed statements are outlined by rectangles in the following listing.

```
100 REM  ***  TEMPERATURE CONVERSIONS
110 REM
120 REM  ***  FUNCTION DEFINITIONS
130 DEF FNC(T)=(T-32)*5/9
140 DEF FNF(T)=(9/5)*T+32
143 REM  ***  DIMENSION OF STRING LENGTHS
145 DIM A$(2,10)
```

```
148 LET A$(1)="FAHRENHEIT"
149 LET A$(2)="CENTIGRADE"
150 REM  ***  USER TEMPERATURE INPUT
160 PRINT \ PRINT "TEMPERATURE";
170 INPUT TO, T$
180 IF T$="C" THEN 270
190 REM  ***  FAHRENHEIT TO CENTIGRADE
200 LET A=FNC(TO)
210 LET K=1
240 REM
250 GOTO 330
260 REM  ***  CENTIGRADE TO FAHRENHEIT
270 LET A=FNF(TO)
280 LET K=2
320 REM  ***  ANSWER PRINT-OUT
330 PRINT TO; "DEGREES ";A$(K); " = ";
340 PRINT A; "DEGREES ";A$(3-K)
350 GOTO 160
360 END
```

Exercise 65. The program to play the game of 23 Matches shown on page 4-9 pits you against the computer. Modify this program so that the computer acts only as a scorekeeper (and referee) and allows you to play 23 Matches with up to 10 other people. Use a string list to store the names of all the players (up to 20 characters each) and print out the name of the player who should move next. The part of your program that accepts and analyzes input might be modeled after lines 220 to 350 of the program on page 4-9. The sample solution to this exercise that appears in Appendix C is modeled after the flowchart which appears in the next column.



COMBINING STRINGS

Sometimes it is convenient to combine two or more strings.

The process of combining strings is known as *concatenation.*Concatenation is indicated by using the ampersand(&).

When strings are concatenated, one is appended to another:

```
10 PRINT "ME" & "YOU"
99 END
RUNNH
MEYOU

READY
```

Concatenation can be used to combine two or more strings into a single variable:

```
10 LET A$="ABCD"
20 LET B$="EFGH"
40 LET D$=A$ & B$
50 PRINT D$
60 LET E$=B$ & A$
70 PRINT E$
99 END

READY
RUNNH
ABCDEFGH
EFGHABCD

READY
```

The length of a concatenated string cannot exceed the maximum length allowed by the system. In the following program, the SL (String too Long) error message was printed because the maximum length of F$ is 8 and the length of *A$ & B$ & C$* is 12:

```
10 LET A$="ABCD"
20 LET B$="EFGH"
30 LET C$="IJKL"
80 LET F$=A$ & B$ & C$
90 PRINT F$
99 END

READY
RUNNH

SL AT LINE 00080

READY
```

The problem can be corrected by adding a DIM statement:

```
5 DIM F$(12)
10 LET A$="ABCD"
20 LET B$="EFGH"
30 LET C$="IJKL"
80 LET F$=A$ & B$ & C$
90 PRINT F$
99 END
RUNNH

ABCDEFGHIJKL

READY
```

Note that *A$ & B$* does not produce the same result as *B$ & A$*. Strings are always combined in the order shown in the statement, and parentheses have no effect:

```
10 LET A$="ABC"
20 LET B$= "DEF"
30 LET C$="GHI"
40 PRINT A$&B$&C$
50 PRINT A$&(B$&C$)
60 PRINT (A$&B$)&C$
99 END

READY
RUNNH
ABCDEFGHI
ABCDEFGHI
ABCDEFGHI

READY
```

Following is a demonstration of combining strings by concatenating first names and last names. The program reads a first name and a last name and then stores them both in a single variable location with the last name first, a comma and a space, and then the first name. Note also the format of the DIM statement at line 10.

```
10 DIM N$(3,20)
20 FOR K=1 TO 3
30 READ F$, L$
40 LET N$(K)=L$ & ", " & F$
50 NEXT K
60 DATA "BOB", "JONES", "RITA", "LAND", "FRANK", "SMITH"
70 FOR K=1 TO 3
80 PRINT N$(K)
90 NEXT K
99 END

READY
RUNNH
JONES, BOB
LAND, RITA
SMITH, FRANK

READY
```

Exercise 66. Use concatenation in a FOR-NEXT loop to create a program that works like this:

?4 HI
HIHIHIHI
?2 LOW
LOWLOW
?

The output should be the result of printing a single variable with a statement such as:

70 PRINT S$

Hint: Look at the INPUT statement in line 170 of the program on page 4-8.

STRING TO NUMERIC CONVERSION

The value (VAL) function. Your work with CLASSIC so far has generally kept strings and numbers separate, even though you know that both types of data may be operated on by the computer. The statement:

40 LET A = A/10

is correct, but:

40 LET A = A$/10

is not. Look at the following program:

```
10 PRINT \ PRINT "YOUR NUMBER"
20 INPUT A$
30 IF A$="QUIT" THEN 99
40 LET A=A$/10
50 LET A=INT(A)+10*(A-INT(A))
60 PRINT "ANSWER= ";A
70 GOTO 10
99 END

READY
RUNNH
MT 40

READY
```

This program results in an MT (Mixed Type) error message because statement 40 tries to perform a numeric operation on a string variable.

The above program is trying to let the user input data for the equation at line 50 and, at the same time, recognize the entry "QUIT" as an indication that the user has entered all the data. The problem is that string and numeric data are stored in the computer in different ways, and the computer cannot work with one where the other is expected. Thus, the statement:

40 LET A = A$/10

is not allowed even though A$ may be "25" or some other number. The way in which data typed in response to an INPUT query will be stored depends upon the type of variable used in the INPUT statement. That is, you can type "25" as a response to either of the following statements:

20 INPUT A

20 INPUT A$

but the first one will store your response as a number and the second will store it as a string.

Strings can be converted to numerics by using the VAL function:

40 LET A = VAL(A$)

The VAL function converts strings to their numeric equivalent.

Here is the corrected program:

```
10 PRINT \  PRINT "YOUR NUMBER";
20 INPUT A$
30 IF A$="QUIT" THEN 99
35 LET A=VAL(A$)
40 LET A=A/10
50 LET A=INT(A)+10*(A-INT(A))
60 PRINT "ANSWER= ";A
70 GOTO 10
99 END

READY
RUNNH

YOUR NUMBER?35
ANSWER=  8

YOUR NUMBER?78
ANSWER=  15

YOUR NUMBER?3
ANSWER=  3

YOUR NUMBER?QUIT

READY
```

Note that the user may now enter both numbers and strings. The VAL function in line 35 converts strings to numerics for use in the equation in line 40.

The above program calculates the sum of the digits in a two-digit number. By modifying this program to recognize the response "HELP", you can make it more meaningful to the user:

```
10 PRINT \ PRINT "YOUR NUMBER";
20 INPUT A$
30 IF A$="QUIT" THEN 99
35 IF A$="HELP" THEN 75
40 LET A=VAL(A$)/10
50 LET A=INT(A)+10*(A-INT(A))
60 PRINT "ANSWER =";A
70 GOTO 10
75 PRINT \ PRINT "THIS PROGRAM WILL COMPUTE THE SUM "
80 PRINT "OF THE DIGITS IN A 2-DIGIT NUMBER. " \ GOTO 10
99 END

READY
RUNNH

YOUR NUMBER?HELP

THIS PROGRAM WILL COMPUTE THE SUM
OF THE DIGITS IN A 2-DIGIT NUMBER.

YOUR NUMBER?42
ANSWER = 6

YOUR NUMBER?QUIT

READY
```

The argument supplied to the VAL function must be a valid string expression; the only operation allowed is concatenation. Following are some experiments with the VAL function to demonstrate how it works with different arguments:

```
10 PRINT VAL(4*5)
99 END
RUNNH
FR 10

READY
```

```
10 PRINT VAL("4*5")
99 END
RUNNH
 4

READY
```

```
10 PRINT VAL("4"*"5")
99 END
RUNNH
MT 10
FR 10

READY
```

```
10 PRINT VAL("4"&"5")
99 END
RUNNH
 45

READY
```

LOOK UP THESE ERROR MESSAGES IN **APPENDIX E OF THE CLASSIC USER'S GUIDE.**

Exercise 67. Use the VAL function to modify the game of 23 Matches on page 4-9 so that it recognizes the response "UNCLE". Program this response to indicate that the human player concedes victory to the computer and wishes to begin the game again with 23 matches.

The string (STR$) function. CLASSIC also has a function that converts numbers to their equivalent strings.

The STR$ function coverts numbers to strings. The resultant strings do not have a leading or trailing blank.

The following two programs demonstrate the difference between numeric and string output:

```
10 FOR K=0 TO 9
20 PRINT K;
30 NEXT K
99 END

READY
RUNNH
 0  1  2  3  4  5  6  7  8  9
READY
```

THIS STATEMENT PRINTS NUMBERS.

```
10 FOR K=0 TO 9
20 PRINT STR$(K);
30 NEXT K
99 END

READY
RUNNH
0123456789
READY
```

THIS STATEMENT PRINTS STRINGS.

Note the presence of the leading and trailing blanks in the first program and their absence in the second.

The STR$ function is very useful in formatting output, especially when used together with the length (LEN) function. The combined use of these functions is discussed below.

OUTPUT FORMATTING

The length (LEN) function. The LEN function allows you to determine the number of characters in a string. Here is a simple program to demonstrate how the LEN function works:

```
10 PRINT LEN("THE QUICK")
20 PRINT LEN("SLY FOX")
30 PRINT LEN("JUMPED OVER THE")
40 PRINT LEN("LAZY BROWN DOG")
99 END


RUNNH
 9
 7
 15
 14

READY
```

Here is another example:

```
10 DIM A$(72)
20 PRINT \ PRINT "YOUR ENTRY";
30 INPUT A$
40 PRINT "  LENGTH =";LEN(A$)
50 GOTO 20
99 END

READY
RUNNH

YOUR ENTRY?NOW IS THE TIME
  LENGTH = 15

YOUR ENTRY?FOR ALL GOOD MEN
  LENGTH = 16

YOUR ENTRY?TO COME TO THE AID OF THEIR COUNTRY.
  LENGTH = 36

YOUR ENTRY?^C
READY
```

The LEN function returns the number of characters in the string indicated in the argument.

With a little work, you can use this function to format numbers as described below.

The format used by CLASSIC to print numeric data causes numbers to be lined up at the left (left-justified) rather than at the right (as is usually done). Look at the following program:

```
10 LET N=1
20 FOR K=1 TO 6
30 PRINT N
40 LET N=N*10
50 NEXT K
99 END

READY
RUNNH
 1
 10
 100
 1000
 10000
 100000

READY
```

To cause these numbers to be lined up at the right (right-justified), you must first know the number of digits in each. Unfortunately, the following expression is **not** allowed:

LEN(N)

because the argument in the LEN function must be a string. Using the STR$ function, you can convert N to a string and then find its length:

5 LET L = LEN(STR$(N))

Following is a FOR-NEXT loop that uses the LEN and STR$ functions in a form similar to that shown above:

```
70 FOR KO=1 TO 6-LEN(STR$(N))
80 PRINT " ";
90 NEXT KO
```

This loop prints a number of blanks depending upon the number of digits in N. If N = 47, then STR$(N) will be "47" and LEN(STR$(N)) = 2. Since 6-2 = 4, this FOR-NEXT loop will be executed 4 times and print 4 blanks.

Exercise 68. Copy the following table onto a separate piece of paper and fill in the blanks.

N	LEN(STR$(N))	6-LEN(STR$(N))
47	2	4
126	____	____
8	____	____
2873	____	____
61045	____	____

Check your work by running the following program:

```
10 READ N
20 PRINT N,LEN(STR$(N)),6-LEN(STR$(N))
30 GOTO 10
40 DATA 47,126,8,2873,61045
99 END

READY
```

The FOR-NEXT loop discussed above was added as a subroutine to the program in column 1 to produce the following results:

```
10 LET N=1
20 FOR K=1 TO 6
25 GOSUB 70
30 PRINT N
40 LET N=N*10
50 NEXT K
60 STOP
70 FOR KO=1 TO 6-LEN(STR$(N))
80 PRINT " ";
90 NEXT KO
95 RETURN
99 END
READY
RUNNH
          1
         10
        100
       1000
      10000
     100000
READY
```

Here is one more example. This time, the above program was modified to line up the decimal points in numeric output:

```
20 FOR K=1 TO 6
23 READ N
25 GOSUB 70
30 PRINT N
50 NEXT K
55 DATA 2.75,333.7,36.74,489.491,5738,92.4725
60 STOP
70 FOR KO=1 TO 4-LEN(STR$(INT(N)))
80 PRINT " ";
90 NEXT KO
95 RETURN
99 END

READY
RUNNH
     2.75
   333.7
    36.74
   489.491
  5738
    92.4725

READY
```

This program used the INT function and considered only the number of digits in the integral part of N. Thus, line 70 used three functions, one inside the next. CLASSIC evaluates each function in turn from the inside out and then uses the result as the argument to the next function. This is called "nesting" functions, and is the same as nesting parentheses when writing numerical expressions.

When functions are nested, you must make sure that the value returned by each function is of the correct type (numeric or string) to be used as the argument for the next function to be called.

Exercise 69. The following program displays a table of the squares and square roots of numbers. Modify this program so that the output is formatted by lining up the decimal points.

```
10 PRINT "N","SQUARE ROOT","FOURTH ROOT"
20 LET N=.1
30 FOR K=1 TO 7
40 PRINT N,
50 PRINT SQR(N),
60 PRINT SQR(SQR(N))
70 LET N=INT(N*10+.5)
80 NEXT K
99 END
RUNNH
N                 SQUARE ROOT     FOURTH ROOT
 0.1              0.316228        0.562341
 1                1               1
 10               3.16228         1.77828
 100              10              3.16228
 1000             31.6228         5.62341
 10000            100             10
 100000           316.228         17.7828

READY
```

Hint: Assign the value to be printed to a temporary variable and use this variable in your subroutine.

The tab (TAB) function. Another function that can be used to format output is the TAB function. This function may only be used in a PRINT statement. Before using it, however, you must understand that CLASSIC allows you to display output in only 72 of the 80 character positions on your screen.

Exercise 70. Enter and run the following program on your CLASSIC system to demonstrate the way in which the columns are numbered on the screen:

```
10 FOR K1=1 TO 8
20 FOR K2=1 TO 9
30 PRINT STR$(K2);
40 NEXT K2
50 PRINT "-";
60 NEXT K1
99 END
```

You will see that after 72 characters are printed, the cursor moves to the beginning of the next line before printing continues.

The argument to the TAB function indicates the column position at which the next character should be printed.

If the cursor is not already at or past the column position indicated by the argument, it is moved to that position before printing continues:

```
10 PRINT TAB(1);"*"
20 PRINT TAB(2);"*"
30 PRINT TAB(3);"*"
99 END

READY
RUNNH
*
 *
  *

READY
```

If the cursor is past the position specified in the argument, the TAB function has no effect.

The next example prints out the column numbers to help you understand the TAB function:

```
10 FOR K1=1 TO 7
20 FOR K2=1 TO 9
30 PRINT STR$(K2);
40 NEXT K2
50 PRINT "-";
60 NEXT K1
70 PRINT "12"
80 PRINT TAB(6); "THIS MESSAGE BEGAN IN COLUMN 6"
90 PRINT TAB(27); "THIS MESSAGE BEGAN IN COLUMN 27"
95 PRINT TAB(42); "THIS MESSAGE BEGAN IN COLUMN 42"
99 END

READY
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
     THIS MESSAGE BEGAN IN COLUMN 6
                          THIS MESSAGE BEGAN IN COLUMN 27
                                         THIS MESSAGE BEGAN IN COLUMN 42

READY
```

By using the TAB function, you can simplify the formatting program that was shown on the previous page. The following program formats integers:

```
10 LET N=1
20 FOR K=1 TO 6
30 PRINT TAB(7-LEN(STR$(N))); N
40 LET N=N*10
50 NEXT K
99 END

READY
RUNNH
      1
     10
    100
   1000
  10000
 100000

READY
```

The next program formats decimals:

```
20 FOR K=1 TO 6
23 READ N
30 PRINT TAB(5-LEN(STR$(INT(N)))); N
50 NEXT K
55 DATA 2.75, 337.3,36.74, 489.491, 5738,92.4725
99 END

READY
RUNNH
    2.75
  337.3
   36.74
  489.491
 5738
   92.4725

READY
```

Line 30 above contains four functions nested one inside the next. Note that each returns the correct *type* of data (string or numeric) required as an argument by the next function to be evaluated.

Exercise 71. Modify the program that you wrote for Exercise 69 so that it uses the TAB function to help format the output. You may wish to define part of the formatting formula in a function to simplify the programming.

The TAB function is extremely useful for drawing graphs. The program below graphs the equation supplied at line 30. (The subroutine at line 80 prints out the column numbers.) Note what happens when the argument to the TAB function is greater than 72.

```
10 GOSUB 80
20 FOR K=1 TO 10
30 LET T=K
40 PRINT TAB(T); "*"
50 NEXT K
60 GOSUB 80
70 STOP
80 FOR K1=1 TO 7 \ FOR K2=1 TO 9 \ PRINT STR$(K2);  \ NEXT K2
90 PRINT "-"; \ NEXT K1 \ PRINT "12" \ RETURN
99 END

READY
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
30 LET T=T+K
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
*
  *
     *
         *
              *
                    *
                           *
                                   *
                                            *
                                                      *
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
30 LET T=K^2
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
*
   *
        *
               *
                        *
                                   *
                                                *
                                                               *

*
*
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
```

Exercise 72. Enter the above program into your workspace and run it for different equations by changing line 30. The following equations will produce some interesting results:

30 LET T=36+((4-K)/2)^ 3

*30 LET T=72*RND(0)*

*30 LET T=2*T+1*

30 LET T=36-(4-K)^ 2

Here is one last example for people interested in math: The program below graphs the sine function. Once again, you can change the statement at line 30 to graph other functions (for example, cosine and tangent).

```
10 LET P=3.14159
20 FOR K=0 TO 4*P STEP P/6
30 LET T=11+10*SIN(K)
40 PRINT TAB(T); "*"
50 NEXT K
99 END

READY
RUNNH
          *
              *
                  *
                   *
                  *
               *
          *
     *
 *
*
 *
    *
         *
              *
                  *
                   *
                  *
               *
          *
     *
 *
*
 *
    *
         *

READY
```

The print (PNT) function. The fourth function that CLASSIC provides to help you format screen output is the PNT function.

The PNT function is used to control the screen through a BASIC language program.

Like the TAB function, the PNT function may only be used in a PRINT statement.

Exercise 73. The following program demonstrates simple screen operations that can be performed with the PNT function. Enter and run this program on your CLASSIC.

```
10 LET A$="BEFORE"
20 LET B$="AFTER"
30 READ N
40 PRINT "N= ";N;A$;PNT(N);B$
50 GOTO 30
60 DATA 7,8,9,10,13
99 END

READY
```

Your results should demonstrate the following actions (line numbers are omitted).

Statement	Action
PRINT PNT(7)	sounds terminal buzzer
PRINT PNT(8)	moves cursor one space to the left
PRINT PNT(9)	moves cursor to next tab stop (tabs stop are in every 8th column: 8,16, 24, etc.)
PRINT PNT(10)	moves cursor down one line
PRINT PNT(13)	moves cursor to left-hand margin of current line

The PRINT statement automatically positions the cursor at the beginning of the following line unless you end the statement line with a semicolon.

Therefore, the following program will print an asterisk at the beginning of a new line instead of in column 8.

```
10 PRINT "HELLO";PNT(9)
20 PRINT "*"
99 END
READY
RUNNH
HELLO
*

READY
```

By adding a semicolon at the end of line 10, the program works as follows:

```
10 PRINT "HELLO";PNT(9);
RUNNH
HELLO*

READY
```

The CLASSIC screen may also be operated in *Escape Mode.* This mode allows certain characters to control the operation of the screen and copier. Under program control, Escape Mode is activated by supplying 27 as the argument to the PNT function:

40 PRINT PNT(27)

The special operation performed is then determined by the next character printed. For example,

40 PRINT PNT(27); "A"

moves the cursor up one line.

Exercise 74. Below is a modification of the program for Exercise 73 to demonstrate the use of the PNT function with Escape Mode. The statements at line 33 and 35 were added just to slow things down enough for you to see what each operation does. Enter and run this program. Respond to the input query by typing any character and pressing RETURN.

```
10 LET A$="BEFORE"
20 LET B$="AFTER"
30 READ N$
33 PRINT "CONTINUE"
35 INPUT Z$
40 PRINT"N$=";N$;"";A$;PNT(27);N$;B$
50 GOTO 30
60 DATA "A","C","H","J","K"
99 END

READY
```

Your results should demonstrate the following actions:

Statement	Action
PRINT PNT(27); "A"	moves cursor up one line
PRINT PNT(27); "C"	moves cursor right one position
PRINT PNT(27); "H"	moves cursor to upper left-hand corner of screen ("home" position)
PRINT PNT(27); "J"	erases from cursor position to end of screen
PRINT PNT(27); "K"	erases line from cursor to right margin

LOOKING BACK

This section has brought you a long way toward understanding some of CLASSIC's more powerful capabilities. It has presented many examples, and hopefully you will see a use for these functions in some of the programs that you plan to write.

Remember these things:

- Undimensioned strings may not exceed 8 characters in length.
- The DIM statement may be used to allow strings up to 72 characters in length. For example:

  10 DIM R$(72)
- String variables may have (at most) one subscript. These variables are dimensioned with the form:

  10 DIM S$(m,n)

  where m is the maximum-valued subscript allowed and n is the maximum length of each string.
- Strings may be concatenated by using the ampersand operator.

The functions presented in this section are summarized below:

LEN	returns the number of characters in a string
PNT	controls special operations on the screen
STR$	converts numeric data to strings
TAB	positions the cursor along a print line
VAL	converts string data to numerics

The next section will help you learn about the remaining six functions available on your CLASSIC system.

SECTION 4-C

STRING AND SPECIAL FUNCTIONS

(Part II)

AUTOMATIC PROGRAM TRACING

You have traced several programs manually to gain an understanding of how specific statements control program flow. Tracing is also valuable for finding bugs in complicated programs. CLASSIC can trace programs automatically by using the TRC function.

The TRC function is used as a switch: it either turns trace mode on or turns it off, depending upon the value of the argument.

TRC(1) turns trace mode on. TRC(0) turns trace mode off.

When trace mode is on, the line number of each statement executed is printed between percent signs (%).

```
10 LET D=TRC(1)
20 LET K=1
30 PRINT "K NOW EQUALS"; K
40 LET K=K+1
45 IF K<=3 THEN30
99 END
```

TURN TRACE ON.

```
READY
RUNNH
% 20 %
% 30 %
K NOW EQUALS 1
% 40 %
% 45 %
% 30 %
K NOW EQUALS 2
% 40 %
% 45 %
% 30 %
K NOW EQUALS 3
% 40 %
% 45 %

READY
```

THE LINE NUMBER OF EACH STATEMENT EXECUTED IS DISPLAYED ENCLOSED IN PERCENT SIGNS AS THE PROGRAM RUNS.

Normal mode is resumed in the following program by turning the trace off with the statement at line 50:

```
10 LET D=TRC(1)
20 LET K=1        <---------- TURN TRACE ON.
30 PRINT "K NOW EQUALS"; K
40 LET K=K+1
45 IF K<=3 THEN30
50 LET D=TRC(0)<---------- TURN TRACE OFF.
55 IF K<=6 THEN 30
99 END

READY
RUN

TRA      BA     3.0      30-DEC-75

% 20 %
% 30 %
K NOW EQUALS 1
% 40 %
% 45 %
% 30 %
K NOW EQUALS 2
% 40 %
% 45 %
% 30 %
K NOW EQUALS 3
% 40 %
% 45 %
% 50 %
K NOW EQUALS 4
K NOW EQUALS 5
K NOW EQUALS 6

READY
```

Even when trace mode is on, the line numbers of some statements are not printed. If you modify the above program by creating a FOR-NEXT loop, the line number of the NEXT statement will not be printed in the trace:

```
10 LET D=TRC(1)
20 FOR K=1 TO 3
30 PRINT "K NOW EQUALS"; K
40 NEXT K
50 LET D=TRC(0)
60 PRINT "END OF PROGRAM"
99 END

READY
RUNNH
% 20 %
% 30 %
K NOW EQUALS 1
% 30 %
K NOW EQUALS 2
% 30 %
K NOW EQUALS 3
% 50 %
END OF PROGRAM

READY
```

Note that the line number of statement 60 is not printed because trace mode is turned off at line 50

The program below prints numbers in ascending order. Note that the line numbers of the GOTO statements (40 and 60) are not printed in the trace.

```
10 LET D=TRC(1)
15 READ A, B
20 IF A>B THEN 50
30 PRINT A; B
40 GOTO 15
50 PRINT B; A
60 GOTO 15
70 DATA 4,7
80 DATA 9,3
90 DATA 5,5
99 END

READY
RUNNH
% 15 %
% 20 %
% 30 %
 4  7
% 15 %
% 20 %
% 50 %
 3  9
% 15 %
% 20 %
% 30 %
 5  5
% 15 %

DA AT LINE 00015

READY
```

The following table lists all the BASIC language statements that are available on CLASSIC and indicates which ones are traced by the TRC function:

Traced	Not Traced
CHAIN	DATA
CLOSE#	DEF
FILE#	DIM
FOR	END
GOSUB	GOTO
IF	NEXT
IF END#	RANDOMIZE
INPUT	REM
INPUT#	STOP
LET	
PRINT	
PRINT#	
READ	
RESTORE	
RESTORE#	
RETURN	

In all of the programs that have been discussed so far, trace mode has been turned on with the statement:

10 LET D = TRC(1)

and turned off with:

50 LET D = TRC(0)

The D in these statements has no meaning; it only serves as a placeholder in the statement *syntax* (grammar). You may use any variable name you choose on the left of the equal sign.

Exercise 76. If you have previously SAVEd a program on a disk, read it into the workspace with the editor OLD command and add the functions TRC(1) and TRC(0) at different places. Run the program to see how this function affects its output. If you have not previously SAVEd a program or would like to write a new one, enter a new program or one that you wrote for a previous exercise into the workspace with the editor NEW command. Include branching statements and several TRC functions. Run your program to see what happens.

GAINING ACCESS TO THE SYSTEM DATE

You learned how to enter the date into your CLASSIC system with the monitor DATE command on page 3-19. You can gain access to this date under program control with the date (DAT$) function.

The DAT$ function returns the system date as an eight-character string.

The dialogue on the next column demonstrates how the DAT$ function works:

```
.DA 2/4/76  <---------- A NEW DATE IS ENTERED.

.DA                                      THE USER
WEDNESDAY FEBRUARY  4, 1976<-- CONFIRMS THE
                                         SYSTEM DATE.
.R BASIC
NEW OR OLD--NEW DATDMO<-- THE USER STARTS UP
                          THE BASIC EDITOR.
READY                                    A NEW
10 PRINT "TODAY'S DATE IS ";DAT$(0)<-- PROGRAM
99 END                                   IS ENTERED.
RUN
                                 THE DAT$ FUNCTION
DATDMO BA  3.0    04-FEB-76<--   RETURNS THE
                                 SYSTEM DATE AS
                                 AN EIGHT CHAR-
TODAY'S DATE IS 02/04/76         ACTER STRING.

READY
```

The argument to the DAT$ function is not significant; it may be any number or numeric expression. If a date has not been entered with the monitor DATE command, the DAT$ function returns an empty string (it has a length of 0).

You can also use the DAT$ function as part of a string expression:

10 LET D$ = DAT$(0)

Since the system date is returned as a string containing only eight characters, the variable on the left side of the equal sign in the above statement does not need to be dimensioned.

Among other uses, the DAT$ function is useful for dating entries in data files. Data files are introduced in Section 4-D.

THE CLASSIC CHARACTER CODE

The character (CHR$) function. Suppose that you wanted to write a program with the backslash (\ ) as part of the print-out. Look what happens:

```
10 PRINT "FIRST PART \ SECOND PART"
99 END
RUNNH
QS 10
LS 10

READY
```

These error messages are printed because CLASSIC interprets line 10 as a multiple-statement line, with two statements separated by the backslash. Neither statement is complete, so two error messages are generated.

Since the backslash causes this problem, CLASSIC provides the character (CHR$) function to reference characters by a special code:

```
10 PRINT "FIRST PART "; CHR$(28); "SECOND PART"
99 END
RUNNH
FIRST PART \SECOND PART

READY
```

Each character that CLASSIC can display has code number between 0 and 63. The backslash is number 28, so the statement:

10 PRINT CHR$(28)

prints the backslash character.

The CHR$ function can also be used to assign characters to a string variable:

```
10 DIM A$(26)
20 FOR K=1 TO 26
30 LET A$=A$ & CHR$(K)
40 NEXT K
50 PRINT A$
99 END

READY
RUNNH
ABCDEFGHIJKLMNOPQRSTUVWXYZ

READY
```

The above program concatenates A$ with each successive character from code number 1 to code number 26. From the print-out for this program, you can see that characters 1 to 26 correspond to the letters of the alphabet.

The decimal code number of each character that can be printed by CLASSIC is shown in the following table:

Decimal	Character	Decimal	Character
0	@	32	(space)
1	A	33	!
2	B	34	"
3	C	35	#
4	D	36	$
5	E	37	%
6	F	38	&
7	G	39	'
8	H	40	(
9	I	41	)
10	J	42	*
11	K	43	+
12	L	44	,
13	M	45	-
14	N	46	.
15	O	47	/
16	P	48	0
17	Q	49	1
18	R	50	2
19	S	51	3
20	T	52	4
21	U	53	5
22	V	54	6
23	W	55	7
24	X	56	8
25	Y	57	9
26	Z	58	:
27	[	59	;
28	\	60	<
29	]	61	=
30	^	62	>
31	_	63	?

Besides providing a method for printing the backslash, the CHR$ function is also useful for understanding the sequence by which CLASSIC sorts string data. This application will be discussed in conjunction with the ASC function.

The ASC function. The ASC function reverses the operation of the CHR$ function.

The ASC function returns the code number of the character supplied as its argument.

The statement:

30 PRINT ASC("E")

will therefore cause the number 5 to be printed.

The program below demonstrates the use of the ASC function to convert characters to their equivalent code numbers. Note that if the argument to the ASC function contains more than one character, for example, "ERIC", the code number of the first character in the string ("E" in this case) is returned.

```
10 PRINT \ PRINT "YOUR CHARACTER";
20 INPUT C$
30 PRINT " "; C$; " IS CHARACTER NUMBER"; ASC(C$)
40 GOTO 10
99 END

READY
RUNNH

YOUR CHARACTER?J
 J IS CHARACTER NUMBER 10

YOUR CHARACTER?H
 H IS CHARACTER NUMBER 8

YOUR CHARACTER?E
 E IS CHARACTER NUMBER 5

YOUR CHARACTER?#
 # IS CHARACTER NUMBER 35

YOUR CHARACTER?\
 \ IS CHARACTER NUMBER 28

YOUR CHARACTER?C
 C IS CHARACTER NUMBER 3

YOUR CHARACTER?^C
READY
```

Sorting string data. The IF statement has been used many times to compare the values of numeric variables. The following program, for example, is a modification of the one that was used on page 4-19 to print two numbers in ascending order:

```
10 PRINT \ PRINT "FIRST NUMBER";
15 INPUT A
18 PRINT "SECOND NUMBER";
20 INPUT B
30 IF A>B THEN 60
40 PRINT "   "; A; "COMES BEFORE"; B
50 GOTO 10
```

continued on next page

```
60 PRINT " ";B; "COMES BEFORE";A
70 GOTO 10
99 END
RUNNH

FIRST NUMBER  ?5
SECOND NUMBER?8
  5 COMES BEFORE 8

FIRST NUMBER  ?12
SECOND NUMBER?6
  6 COMES BEFORE 12

FIRST NUMBER  ?^C
READY
```

Here is a second modification of the program to allow it to compare strings:

```
LISTNH
10 PRINT \ PRINT "FIRST LETTER";
15 INPUT A$
18 PRINT "SECOND LETTER";
20 INPUT B$
30 IF A$>B$ THEN 60
40 PRINT "  ";A$;" COMES BEFORE ";B$
50 GOTO 10
60 PRINT "  ";B$;" COMES BEFORE ";A$
70 GOTO 10
99 END
;
READY
RUNNH

FIRST LETTER?A
SECOND LETTER?J
  A COMES BEFORE J

FIRST LETTER?&
SECOND LETTER?#
  # COMES BEFORE &

FIRST LETTER?^C
READY
```

In the first case above, A comes before J because A is character number 1 and J is character number 10. In the second case, # comes before & because # is character number 35 and & is character number 38.

Look what happens when you compare strings that are two to eight characters in length:

```
10 PRINT \ PRINT "FIRST NAME";
15 INPUT A$
18 PRINT "SECOND NAME";
20 INPUT B$
30 IF A$>B$ THEN 60
40 PRINT "  "; A$; " COMES BEFORE "; B$
50 GOTO 10
60 PRINT "  "; B$; " COMES BEFORE "; A$
70 GOTO 10
99 END

READY
RUNNH

FIRST NAME?JOSEPH
SECOND NAME?MARY
  JOSEPH COMES BEFORE MARY

FIRST NAME?MOSES
SECOND NAME?AARON
  AARON COMES BEFORE MOSES

FIRST NAME?ABRAHAM
SECOND NAME?ISAAC
  ABRAHAM COMES BEFORE ISAAC

FIRST NAME?ART
SECOND NAME?ARTHUR
  ARTHUR COMES BEFORE ART

FIRST NAME?BILLY
SECOND NAME?BILL
  BILLY COMES BEFORE BILL

FIRST NAME?^C
READY
```

Here CLASSIC makes the decision as to which string is greater by comparing the two first characters. If these characters are the same, the two second characters are compared. If these are the same, the two third characters are compared, and so on. For example, when comparing JOSEPH to JOHN, the decision as to which is greater is made after the third pair of characters:

```
J   O   S   E   P   H
↕   ↕   ↕   ↕
J   O   H   N
```

JOHN comes before JOSEPH ("JOHN" < "JOSEPH") because H is character number 8 and S is character number 19.

Look at the last two comparisons in the previous program. CLASSIC reported that ARTHUR comes before ART and BILLY comes before BILL. This order is not consistent with the rules that most people follow when putting names in alphabetical order. In the phone book, for example, ART would come before ARTHUR.

This comparison problem is caused by the fact that CLASSIC ran out of letters in one of the strings before a decision could be made:

```
A   R   T   H   U   R
↕   ↕   ↕
A   R   T   ?
```

When this happens, CLASSIC concatenates the shorter string with spaces until it is the same length as the longer string:

```
A   R   T   H   U   R
↕   ↕   ↕   ↕   ↕   ↕
A   R   T   ␣   ␣   ␣
```

("␣" represents a space.) CLASSIC therefore decides that ARTHUR comes before ART because H (character number 8) comes before space (character number 32).

You saw a program to sort numbers on page 3-36. The program on the following page performs the same operation with strings. That is, it arranges the elements in a one-dimensional string array in ascending order by successive comparisons. This program uses nested loops with K1 and K2 as the indices of the two loops. The comparison is done at line 240. If A$(K1) ≤ A$(K2), the program increments K2 and another comparison is made. But if A$(K1) > A$(K2), the string in A$(K1) is switched with the string in A$(K2) by the statements at lines 250-270 before incrementing K2.

```
100 FOR K=1 TO 5 \ READ A$(K) \ NEXT K
110 DATA 'BE','BEE','BEET','BEETS','BEETLE'
```

SORT ROUTINE

```
120 FOR K1=1 TO 4
130 FOR K2=K1+1 TO 5
240 IF A$(K1)<A$(K2) THEN 280
250 LET T$=A$(K1)
260 LET A$(K1)=A$(K2)
270 LET A$(K2)=T$
280 NEXT K2
290 NEXT K1
```

```
300 PRINT \ PRINT "SORTED DATA:"\ PRINT
310 FOR K=1 TO 5
320 PRINT A$(K)
330 NEXT K
999 END

READY
RUNNH

SORTED DATA:

BEETLE
BEETS
BEET
BEE
BE

READY
```

Note the form of the two FOR statements in this program:

120 FOR K1=1 TO 4
130 FOR K2=K1+1 TO 5

This arrangement makes the maximum number of comparisons that are ever needed to sort any five pieces of data with the routine used in this program. This type of sorting routine is called a bubble sort because the smaller values are "bubbled" up to the top of the list in a stepwise manner.

Since CLASSIC concatenates shorter strings with spaces before a comparison is made, the results of the above program were not printed in alphabetical order as you would find them in the dictionary. You can modify this procedure by concatenating shorter strings with the at sign (@) before the values of the strings are compared. Since the @ sign is character number 0, this action will cause the value of the shorter string to be less than the value of the longer string if the corresponding leading characters are the same.

B E @

↕ ↕ ↕

B E E

BE @ comes before BEE because @ comes before E. The following program demonstrates concatenation with the at sign.

```
100 FOR K=1 TO 5 \ READ A$(K) \ NEXT K
110 DATA 'BE','BEE','BEET','BEETS','BEETLE'
120 FOR K1=1 TO 4
130 FOR K2=K1+1 TO 5
```

NEW STATEMENTS TO CONCATENATE SHORTER STRINGS WITH THE AT (@) SIGN BEFORE COMPARISON.

```
140 IF LEN(A$(K1))=LEN(A$(K2)) THEN 240
150 LET X$=A$(K1)
155 LET Y$=A$(K2)
160 IF LEN(X$)>LEN(Y$) THEN 210
170 FOR K=1 TO LEN(Y$)-LEN(X$)
180 LET X$=X$ & '@'
190 NEXT K
200 GOTO 230
210 FOR K=1 TO LEN(X$)-LEN(Y$)
220 LET Y$=Y$ & '@'
225 NEXT K
230 IF X$<Y$ THEN 280
235 GOTO 250
```

```
240 IF A$(K1)<A$(K2) THEN 280
250 LET T$=A$(K1)
260 LET A$(K1)=A$(K2)
270 LET A$(K2)=T$
280 NEXT K2
290 NEXT K1
300 PRINT \ PRINT 'SORTED DATA:'\ PRINT
310 FOR K=1 TO 5
320 PRINT A$(K)
330 NEXT K
999 END

READY
RUNNH

SORTED DATA:

BE
BEE
BEET
BEETLE
BEETS

READY
```

In this program, the lengths of A$(K1) and A$(K2) are compared at line 140. If they are of equal lengths, the program branches to line 240 where a normal comparison is made. (Line 240 is exactly the same as it was in the previous program.) But if the strings are of unequal length, they are first stored in temporary variables (X$ and Y$ — see lines 150 and 155). The program then determines which is shorter (line 160), and the shorter string is concatenated with @ signs until it is the same length as the longer string (lines 170-190 and 210-225). A comparison is then made between the modified strings (line 230). If the relation specified in line 230 is true, the program branches to line 280, K2 is incremented, and the loop is repeated. If the relation is false, the program goes to line 250 and the values of A$(K1) and A$(K2) are switched.

Exercise 77. Modify the above program to sort up to 100 strings, each 20 characters in length. Indicate the number of strings to be sorted as the first item in your data table. Use a DIM statement to dimension your string list and modify the FOR statements to handle a variable number of data items.

TAKING STRINGS APART

On page 4-10 you learned how to put strings together by concatenation. The last two functions that CLASSIC provides will allow you to take strings apart.

The position (POS) function. The position function is used to search one string to find out if another string is contained within it. This function takes three arguments:

```
               POS(A$,T$,N)
function name ———┘  ↑  ↑  ↑
string to be searched┘  |  |
string to be searched for┘  |
position at which to begin search
```

The function POS(A$,T$,N) searches string A$ for the first occurrence of T$ starting at position N.

If T$ is part of A$, the POS function returns the number of the position at which the first character in T$ occurs in A$. If T$ is not in A$, the POS function returns the value 0.

Look at the following example:

```
10 DIM A$(52)
15 FOR J=1 TO 2
20 FOR K=1 TO 26 \ LET A$=A$ & CHR$(K) \ NEXT K
25 NEXT J
30 PRINT \ FOR K=1 TO 5 \ PRINT '123456789-'; \NEXT K
35 PRINT '12'
40 PRINT A$
50 PRINT POS(A$, 'N', 1)
60 PRINT POS(A$, 'NO', 1)
70 PRINT POS(A$, 'N', 20)
80 PRINT POS(A$, 'XYZ', 26)
90 PRINT POS(A$, 'FED', 1)
99 END


RUNNH

123456789-123456789-123456789-123456789-123456789-12
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
 14
 14
 40
 50
 0

READY
```

Notice that the value 14 is printed by line 60 as well as line 50 because the POS function returns the number of the position at which the **first** character of the string to be searched for ("N" or "NO") occurs. The value 40 was printed by line 70 because the search for N started at position 20 rather than position 1. Line 90 printed 0 because the string FED was not found at all.

Exercise 78. Enter the following program to the computer and use it to experiment with the POS function as shown in the sample run.

```
10 DIM A$(26)
20 FOR K=1 TO 26 \ LET A$=A$ & CHR$(K) \ NEXT K
30 PRINT \ PRINT '123456789-123456789-123456'
40 PRINT A$
50 PRINT \ PRINT 'WHAT LETTER DO YOU NEED';
60 INPUT T$
65 LET P=POS(A$,T$,1)
70 PRINT ' '; T$; ' IS AT POSITION';P
80 GOTO 50
99 END

READY
RUNNH

123456789-123456789-123456
ABCDEFGHIJKLMNOPQRSTUVWXYZ

WHAT LETTER DO YOU NEED?E
 E IS AT POSITION 5

WHAT LETTER DO YOU NEED?V
 V IS AT POSITION 22

WHAT LETTER DO YOU NEED?JKL
 JKL IS AT POSITION 10

WHAT LETTER DO YOU NEED?#
 # IS AT POSITION 0

WHAT LETTER DO YOU NEED?^C
READY
```

The next set of examples will examine a use of the POS function to search for a "key" set of characters in a user entry.

Begin by studying the program below. This program presents the user with two one-digit numbers and asks him or her to enter the sum.

```
110 PRINT \ PRINT 'THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.'
120 RANDOMIZE
130 LET A=INT(10*RND(0))
140 LET B=INT(10*RND(0))
150 PRINT \ PRINT 'HOW MUCH IS'; A; '+'; B;
160 INPUT C
180 IF C=A+B THEN 230
210 PRINT '   INCORRECT.  PLEASE TRY AGAIN...'
220 GOTO 150
230 PRINT ' CORRECT!'
240 GOTO 130
270 END

READY
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 3 + 7 ?10
 CORRECT!

HOW MUCH IS 6 + 9 ?12
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 6 + 9 ?15
 CORRECT!

HOW MUCH IS 4 + 8 ?^C
READY
```

Suppose the user decided to enter "IT'S 3" instead of just "3" when asked for the sum of 3 and 0. Look what would happen:

```
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 8 + 8 ?IT'S 16
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 8 + 8    INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 8 + 8    INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 8 + 8    INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 8 + 8  CORRECT!

HOW MUCH IS 8 + 3 ?^C
READY
```

Each character in the entry "IT'S" is interpreted by the system as a 0. Since there are four characters (I,T,', and S), the system prints the "incorrect" message four times before it finally reaches the 3 and judges the answer "correct".

A modified version of this program that corrects the above problem is shown on the next page. This version uses the POS function to search the user's entry for the correct answer. The new statements have been enclosed in boxes.

```
100 DIM A$(72)
110 PRINT \ PRINT 'THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.'
120 RANDOMIZE
130 LET A=INT(10*RND(0))
140 LET B=INT(10*RND(0))
150 PRINT \ PRINT 'HOW MUCH IS'; A; '+'; B;
160 INPUT A$
170 LET C$=STR$(A+B)
180 IF POS(A$, 'QUIT',1)>0 THEN 270
190 IF POS(A$, 'HELP',1)>0 THEN 250
200 IF POS(A$,C$,1)>0 THEN 230
210 PRINT '   INCORRECT.  PLEASE TRY AGAIN...'
220 GOTO 150
230 PRINT ' CORRECT!'
240 GOTO 130
250 PRINT ' '; A; '+'; B; '=';A+B; '. HERE'S ANOTHER...'
260 GOTO 130
270 END

READY
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.
```

```
HOW MUCH IS 7 + 0 ?IT'S 7
 CORRECT!

HOW MUCH IS 7 + 6 ?THE ANSWER IS 13
 CORRECT!

HOW MUCH IS 7 + 2 ?UM...8?
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 7 + 2 ?WOULD YOU BELIEVE, 9
 CORRECT!

HOW MUCH IS 3 + 8 ?I NEED A LITTLE HELP
  3 + 8 = 11 . HERE'S ANOTHER...

HOW MUCH IS 1 + 3 ?GOSH, I KNOW THAT'S 4!!!
 CORRECT!

HOW MUCH IS 8 + 4 ?WON'T YOU EVER QUIT?

READY
```

In order to use the POS function, both the user's response and the correct answer had to be stored as strings (see lines 160 and 170). Once this was done, it was also possible to search for the words "QUIT" and "HELP" (lines 180 and 190). This type of response decoding is called a **keyword search.**

Note the form of the IF statement at line 180:

180 IF POS(A$,"QUIT",1) > 0 THEN 270

Remember that the POS function returns a positive integer if the second string is found in the first, and a value of 0 if it is not. This IF statement will therefore cause a branch if and only if "QUIT" is in A$.

Exercise 79. Enter the above program and see if you can fool this program by making it think that an incorrect answer is correct.

The segment (SEG$) function. Here is one way that the arithmetic program can be fooled:

```
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 9 + 9 ?18
 CORRECT!

HOW MUCH IS 7 + 3 ?-10
 CORRECT!

HOW MUCH IS 6 + 1 ?^C
READY
```

In the second problem (3+1), the program searched the user's response for a 4. It found a 4 and judged the answer to be correct even though the actual entry was **negative** 4. By using the SEG$ function, you can make the program sophisticated enough to distinguish between positive 4 and negative 4 even when working with strings.

The SEG$ function returns a segment of the string specified in its argument.

The following program demonstrates how the SEG$ function works:

```
10 DIM A$(26)
20 FOR K=1 TO 26 \ LET A$=A$ & CHR$(K) \ NEXT K
30 PRINT "123456789-123456789-123456"
40 PRINT A$
50 PRINT SEG$(A$,2,6)
60 PRINT SEG$(A$,13,24)
70 PRINT SEG$(A$,4,10)
80 PRINT SEG$(A$,21,21)
99 END
```

```
READY
RUNNH
123456789-123456789-123456
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEF
MNOPQRSTUVWX
DEFGHIJ
U

READY
```

The program returns segments of the string A$ (which contains the 26 letters of the alphabet). The complete string and the number of each position are first printed by lines 30 and 40. Lines 50 through 80 then print segments of this string.

Like the POS function, the SEG$ function requires three arguments:

SEG$ (A$,X,Y)

function name
string to be segmented
position of first character
position of last character

The function SEG$ (A$,X,Y) returns the Xth through Yth characters in A$ inclusive.

Exercise 80. Enter the program below into your computer and use it to experiment with the SEG$ function as in the sample run shown.

On page 4-26 is a modification of the arithmetic program which uses the SEG$ function to catch negative inputs. The modified statements are enclosed in a box. Note that both numeric arguments to the SEG$ function are the same in this case (P-1), because the program only needs to compare a single character.

Here is another way to trick the arithmetic program:

```
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 7 + 4 ?THE ANSWER IS NOT 11
 CORRECT!
```

```
10 DIM A$(26)
20 FOR K=1 TO 26 \ LET A$=A$ & CHR$(K) \ NEXT K
30 PRINT \ PRINT "123456789-123456789-123456"
40 PRINT A$
50 PRINT \ PRINT "WHICH LETTERS WOULD YOU LIKE";
60 INPUT X,Y
70 PRINT TAB(X); SEG$(A$,X,Y)
80 GOTO 50
99 END
```

```
READY
RUNNH

123456789-123456789-123456                    THIS IS THE STRING TO BE
ABCDEFGHIJKLMNOPQRSTUVWXYZ                    SEGMENTED.

WHICH LETTERS WOULD YOU LIKE?4,12             THE 4TH THROUGH 12TH
   DEFGHIJKL                                  CHARACTERS ARE RETURN-
                                              ED.

WHICH LETTERS WOULD YOU LIKE?18,22            THE 18TH THROUGH 22ND
                  RSTUV                       CHARACTERS ARE RETURN-
                                              ED.

WHICH LETTERS WOULD YOU LIKE?7,14
      GHIJKLMN

WHICH LETTERS WOULD YOU LIKE?11,11            X=Y SO ONLY ONE CHAR-
          K                                   ACTER IS RETURNED.

WHICH LETTERS WOULD YOU LIKE?10,30            Y   LEN(X$) SO Y IS SET TO
         JKLMNOPQRSTUVWXYZ                    LEN (X$), or 28.

WHICH LETTERS WOULD YOU LIKE?16,8             X   Y SO NO CHARACTERS
                                              ARE RETURNED.

WHICH LETTERS WOULD YOU LIKE?1,5              WHEN X=0 IT IS SET TO 1 BY
ABCDE                                         THE STYSTEM.

WHICH LETTERS WOULD YOU LIKE?0,5              IF X OR Y IS NEGATIVE, AN
ABCDE                                         ERROR MESSAGE IS PRINT-
                                              ED AND THE PROGRAM
                                              STOPS.

WHICH LETTERS WOULD YOU LIKE?-1,5

FM AT LINE 00070

READY
```

```
100 DIM A$(72)
110 PRINT \ PRINT 'THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.'
120 RANDOMIZE
130 LET A=INT(10*RND(0))
140 LET B=INT(10*RND(0))
150 PRINT \ PRINT 'HOW MUCH IS'; A; '+'; B;
160 INPUT A$
170 LET C$=STR$(A+B)
180 IF POS(A$, 'QUIT',1)>0 THEN 270
190 IF POS(A$, 'HELP', 1)>0 THEN 250
200 IF POS(A$,C$,1)>0 THEN 230
210 PRINT '   INCORRECT.  PLEASE TRY AGAIN...'
220 GOTO 150
230 LET P=POS(A$,C$,1)
232 IF SEG$(A$,P-1,P-1)<>'-' THEN 236
234 GOTO 210
236 PRINT ' CORRECT!'
240 GOTO 130
250 PRINT ' ';A; '+';B;'=';A+B;'.  HERE'S ANOTHER...'
260 GOTO 130
270 END

READY
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 2 + 9 ?-11
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 2 + 9 ?11
 CORRECT!

HOW MUCH IS 5 + 1 ?-6
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 5 + 1 ?6
 CORRECT!

HOW MUCH IS 9 + 0 ?HELP
  9 + 0 = 9 .  HERE'S ANOTHER...

HOW MUCH IS 2 + 1 ?LET'S QUIT FOR NOW

READY
```

This problem can be corrected by adding the following statements:

```
236 IF POS(A$, 'NOT',1)=0 THEN 238
237 IF POS(A$,'NOT',1)<POS(A$,C$,1)THEN 210
238 PRINT ' CORRECT!'
```

The run below demonstrates this improvement, but also turns up another weakness, failure to recognize "N'T".

```
THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 0 + 4 ?THE ANSWER IS NOT 4
   INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 0 + 4 ?THE ANSWER IS 4???
 CORRECT!

HOW MUCH IS 1 + 7 ?THAT CAN'T BE 8
 CORRECT!

HOW MUCH IS 2 + 5 ?WOW,WE BETTER QUIT

READY
```

Exercise 81. Try to fix this problem yourself. (You may have to resequence the program to get more room). Run your new version and find still more ways to increase the program's ability to detect incorrect answers.

Exercise 82. For a real challenge, try to fix the following problem:

```
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 9 + 6 ?0123456789101112131415161718
 CORRECT!

HOW MUCH IS 9 + 0 ?0123456789101112131415161718
 CORRECT!

HOW MUCH IS 7 + 8 ?0123456789101112131415161718
 CORRECT!

HOW MUCH IS 8 + 7 ?^C
READY
```

Hint: Use the POS function to find the correct answer. Then use the SEG$ function to check the positions before and after the correct answer (if any). You can find out if the characters in these two positions are numbers by examining their codes using the ASC function to see if they fall between 48 and 57.

LOOKING BACK

In this section, you have studied the last six functions that are available in CLASSIC BASIC:

ASC(X$)	returns the code number of the first character in X$
CHR$(N)	returns the character whose code number is N
DAT$(0)	returns the system date (if any) in the form MM/DD/YY

POS(A$,T$,N)	searches A$ for the first occurrence of T$ starting at position N
SEG$(A$,X,Y)	returns a segment of A$ from positions X to Y
TRC(N)	turns trace mode on if N=1 and turns trace mode off if N=0

Perhaps you have noticed the following rule:

Functions whose names end in a dollar sign ($) always return strings. All other functions return numbers.

Chapter 5 in the *CLASSIC User's Reference Guide* summarizes all of the BASIC functions that are available on the CLASSIC system and provides a ready reference for your future use. For more examples of function usage, look at the listings of the programs supplied in Appendix A.

The next section will introduce you to the use of disk files for storing data and help you learn the remaining seven statements that can be used in a BASIC language program on CLASSIC.

SECTION 4-D

STORING DATA IN DISK FILES

PROGRAM CHAINING

It is possible to write a BASIC program so large that CLASSIC will not be able to run it. When this happens, you will get the TB (Too Big) error message. The easiest way to correct this problem is usually to break the program into two parts and then **chain** from one program to the other. Chaining is performed with the BASIC CHAIN statement. For example:

30 CHAIN "RXA1:TARGET.BA"

The CHAIN statement causes the program specified to be run.

The general format of the CHAIN statement is:

line number CHAIN "dev:filnam.ex"

The complete device, file name, and extension of the program to be run should be specified, as no default parameters are assumed by the system. No matter how many programs are chained to each other, the workspace will always contain the first program in the chain when control finally returns to the editor.

A simple use of the CHAIN statement is demonstrated at the right.

Besides allowing programs to be of virtually unlimited size, the CHAIN statement can also be used to create a master control program for a set of computer programs. The program on the next page prints out a list of all the programs on the BASIC Program Demonstration Disk and allows you to chain to a program simply by entering a file name.

Look at the format of the CHAIN statement at line 420. Since a string expression is used rather than a simple string, "RXA1:" and ".BA" have to be concatenated with A$ to complete the dev:filnam.ex form required as the parameter of the CHAIN statement. Note once again that the workspace contains the first program in the chain when control finally returns to the editor. Therefore, RUNNH causes the index program to be rerun. The CL error message results because file RXA1:GEUSS.BA could not be found.

To prevent the CL message, you can modify the index program as shown on page 4-29. (The new statements have been enclosed in a box). This program simply checks the validity of the user's entry by comparing it to each available program name in turn (see lines 280-310). When a match is found, the CHAIN statement at line 420 is executed. If no match is found, a message is printed and the user is asked to make a new entry (lines 350-380).

Exercise 83. Obtain a copy of the BASIC Program Demonstration Disk from the person in charge of your CLASSIC (your system manager) and enter the program on page 4-29. Use it to CHAIN to various BASIC demonstration programs.

```
.R BASIC
NEW OR OLD--NEW TARGET
```

A NEW PROGRAM CALLED **TARGET** IS ENTERED INTO THE WORKSPACE.

```
READY
10 PRINT "THIS MESSAGE IS BEING PRINTED BY THE ";
20 PRINT "TARGET PROGRAM."
99 END
RUN

TARGET  BA    3.0

THIS MESSAGE IS BEING PRINTED BY THE TARGET PROGRAM.
```

WHEN RUN, **TARGET** PRINTS A MESSAGE THAT IDENTIFIES ITSELF.

```
READY
SAVE RXA1:TARGET
```

TARGET IS SAVED ON RXA1.

```
READY
NAME CHAIN
```

THE WORKSPACE NAME IS CHANGED TO **CHAIN.**

```
READY
20 PRINT "CHAINING PROGRAM"
LIST
```

STATEMENT 20 IS CHANGED AND THE MODIFIED PROGRAM IS LISTED.

```
CHAIN   BA    3.0

10 PRINT "THIS MESSAGE IS BEING PRINTED BY THE ";
20 PRINT "CHAINING PROGRAM"
99 END

READY
RUN

CHAIN   BA    3.0

THIS MESSAGE IS BEING PRINTED BY THE CHAINING PROGRAM
```

WHEN RUN, **CHAIN** ALSO PRINTS A MESSAGE THAT IDENTIFIES ITSELF.

```
READY
30 CHAIN "RXA1:TARGET.BA"
LIST
```

A CHAIN STATEMENT IS ADDED AS LINE 30 AND THE CONTENTS OF THE WORKSPACE ARE LISTED AGAIN.

```
CHAIN   BA    3.0

10 PRINT "THIS MESSAGE IS BEING PRINTED BY THE ";
20 PRINT "CHAINING PROGRAM"
30 CHAIN "RXA1:TARGET.BA"
99 END

READY
RUN

CHAIN   BA    3.0

THIS MESSAGE IS BEING PRINTED BY THE CHAINING PROGRAM
THIS MESSAGE IS BEING PRINTED BY THE TARGET PROGRAM.
```

WHEN THE MODIFIED PROGRAM IS RUN, IT CHAINS TO **TARGET**. THEREFORE, THE IDENTIFYING MESSAGES ARE PRINTED BY BOTH PROGRAMS.

```
READY
LIST

CHAIN   BA    3.0

10 PRINT "THIS MESSAGE IS BEING PRINTED BY THE ";
20 PRINT "CHAINING PROGRAM"
30 CHAIN "RXA1:TARGET.BA"
99 END

READY
```

WHEN CONTROL RETURNS TO THE EDITOR, THE WORKSPACE CONTAINS THE FIRST PROGRAM IN THE CHAIN.

```
100 PRINT \ PRINT "THE PROGRAMS AVAILABLE ON THE BASIC"
110 PRINT "PROGRAM DEMONSTRATION DISK ARE:" \ PRINT
120 REM
130 REM  ***  PROGRAM NAME PRINTER
140 REM
150 FOR K=1 TO 6
160 READ N1$,N2$,N3$
170 PRINT N1$,N2$,N3$
180 NEXT K
190 REM
200 REM  ***  INPUT QUERY
210 REM
220 PRINT \ PRINT "WHICH WOULD YOU LIKE TO RUN";
230 INPUT A$ \ PRINT
390 REM
400 REM  *** CHAIN STATEMENT
410 REM
420 CHAIN "RXA1:" & A$ & ".BA"
430 REM
440 REM  *** DATA TABLE
450 REM
460 DATA "ACEY02","ATTEND","ATTSET", "CALC","EASY02"
470 DATA "EASY03","GUESS","HMRABI","HURKLE","HURK02"
480 DATA "MORGAG","QUADEQ","QUAD02","QUAD03","SYNONY"
490 DATA "SYNSET","WTDAVG",""
500 END

READY
RUNNH

THE PROGRAMS AVAILABLE ON THE BASIC
PROGRAM DEMONSTRATION DISK ARE:

ACEY02          ATTEND          ATTSET
CALC            EASY02          EASY03
GUESS           HMRABI          HURKLE
HURK02          MORGAG          QUADEQ
QUAD02          QUAD03          SYNONY
SYNSET          WTDAVG

WHICH WOULD YOU LIKE TO RUN?CALC


YOUR EXPRESSION?45*6

45*6 = 270

YOUR EXPRESSION?QUIT


READY
RUNNH

THE PROGRAMS AVAILABLE ON THE BASIC
PROGRAM DEMONSTRATION DISK ARE:

ACEY02          ATTEND          ATTSET
CALC            EASY02          EASY03
GUESS           HMRABI          HURKLE
HURK02          MORGAG          QUADEQ
QUAD02          QUAD03          SYNONY
SYNSET          WTDAVG

WHICH WOULD YOU LIKE TO RUN?GEUSS


CL AT LINE 00420

READY
```

```
100 PRINT \ PRINT "THE PROGRAMS AVAILABLE ON THE BASIC"
110 PRINT "PROGRAM DEMONSTRATION DISK ARE:" \ PRINT
120 REM
130 REM  ***  PROGRAM NAME PRINTER
140 REM
150 FOR K=1 TO 6
160 READ N1$,N2$,N3$
170 PRINT N1$,N2$,N3$
180 NEXT K
190 REM
200 REM  ***  INPUT QUERY
210 REM
220 PRINT \ PRINT "WHICH WOULD YOU LIKE TO RUN";
230 INPUT A$ \ PRINT
240 REM
250 REM  ***  CHECK FOR VALID PROGRAM NAME
260 REM
270 RESTORE
280 FOR K=1 TO 17
290 READ N$
300 IF N$=A$ THEN 420
310 NEXT K
320 REM
330 REM  ***  MESSAGE FOR INVALID NAME
340 REM
350 PRINT A$; " IS NOT ON THE DEMONSTRATION DISK."
360 RESTORE
370 PRINT "CHOOSE ANOTHER " \ PRINT
380 GOTO 150
390 REM
400 REM  *** CHAIN STATEMENT
410 REM
420 CHAIN "RXA1:" & A$ & ".BA"
430 REM
440 REM  *** DATA TABLE
450 REM
460 DATA "ACEY02","ATTEND","ATTSET", "CALC","EASY02"
470 DATA "EASY03","GUESS","HMRABI","HURKLE","HURK02"
480 DATA "MORGAG","QUADEQ","QUAD02","QUAD03","SYNONY"
490 DATA "SYNSET","WTDAVG",""
500 END

READY
RUNNH

THE PROGRAMS AVAILABLE ON THE BASIC
PROGRAM DEMONSTRATION DISK ARE:

ACEY02          ATTEND          ATTSET
CALC            EASY02          EASY03
GUESS           HMRABI          HURKLE
HURK02          MORGAG          QUADEQ
QUAD02          QUAD03          SYNONY
SYNSET          WTDAVG

WHICH WOULD YOU LIKE TO RUN?GEUSS

GEUSS IS NOT ON THE DEMONSTRATION DISK.
CHOOSE ANOTHER

ACEY02          ATTEND          ATTSET
CALC            EASY02          EASY03
GUESS           HMRABI          HURKLE
HURK02          MORGAG          QUADEQ
QUAD02          QUAD03          SYNONY
SYNSET          WTDAVG

WHICH WOULD YOU LIKE TO RUN?GUESS

GUESS:  THE NUMBER GUESSING GAME
--------------------------------

PLEASE TYPE YOUR FIRST NAME AND THEN PRESS THE
RETURN KEY.

WHAT IS YOUR FIRST NAME?
```

Exercise 84. Write two programs of your own that chain to each other. Make each print out at least one message and require at least one user entry. Note the amount of time that it takes one program to chain to the other, especially if you write large programs. In Section 4-E you will learn a way to speed up the chaining process.

STRING DATA FILES

So far, you have used the CLASSIC disk only to store BASIC language programs. However, disk files can also contain data for use by BASIC programs. Storing data in disk files is one of the most powerful uses of a computer system because it allows a program to work with an extremely large amount of data in a minimum amount of time. Disk files also allow data generated as output by one program to be used as input to another program without requiring you to enter it manually through the keyboard.

Writing a disk file. Data is written to a disk file in very much the same manner as it is written to the screen. That is, you simply have to tell the system the name of the file that you want the data written into and then use a variation of the PRINT statement to do the actual writing.

Specifying the name of a file to be written (or read) is known as **opening a file.** To open a file, use the FILE# statement. This statement tells the system the type of file that you wish to open, the number by which you will refer to the file in your program, and the name of the file. For example,

10 FILEV #1: "RXA1:OUTPUT.JH"

file type
file number
file name

File type V is used for all files that will receive string output. The number that is assigned to the file in the FILE# statement is used by the PRINT# statement to indicate where data will be written:

20 PRINT #1: "THIS DATA IS STORED IN A DISK FILE."

The above statement causes the message indicated to be written into the file opened as file number 1.

Whenever data is written to a file, that file must be closed before program execution stops or the file itself will be erased:

30 CLOSE #1

When these three statements are put together into a program, here is what happens:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 PRINT #1:  "THIS DATA IS STORED IN A DISK FILE."
30 CLOSE #1
99 END

READY
RUNNH


READY
```

This program does not cause anything to be output to the screen. However, you can verify that something has actually been written to the disk file by using the monitor TYPE command:

```
BYE

.TYPE RXA1:OUTPUT.JH
THIS DATA IS STORED IN A DISK FILE.
```

In addition, the name of the file written will appear in the disk directory.

The following rules apply to use of the FILE#, PRINT#, and CLOSE# statements:

- In all three statements, the file number is preceded by a number sign (#).
- In the FILE# and PRINT# statements, the file number is followed by a colon (:).
- A colon is not used in the CLOSE# statement.
- The file number may be any numeric expression whose value is between 0 and 4, inclusive.
- The file name in a FILE# statement may be any string expression, but all parameters must be specified in the form dev:filnam.ex and enclosed in quotes. No defaults are assumed.
- File #0 is always open and always refers to the keyboard/screen as shown in the following example:

```
10 PRINT #0:  "OUTPUT TO SCREEN"
99 END

READY
RUNNH
OUTPUT TO SCREEN

READY
```

- Except for the addition of the file number, the PRINT# statement (when used with alphanumeric files) works just like the PRINT statement. That is, the exact same rules regarding output format (spacing, print zones, use of commas, semicolons, the TAB and PNT functions, etc.) are followed for both statements. The PRINT# statement simply creates a disk file containing information just as it would be printed on the screen.

Reading a disk file. To read from a disk file, you must first open it as an input file. If the file contains string data, specify **no** file type in the FILE statement:

50 FILE #1: "RXA1:OUTPUT.JH"

Once the file is opened for reading, data is read with the INPUT# statement:

60 INPUT #1: A$

These statements are demonstrated in the following program:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 PRINT #1:  "THIS DATA IS STORED IN A DISK FILE."
30 CLOSE #1
40 DIM A$(72)
50 FILE #1: "RXA1:OUTPUT.JH"
60 INPUT #1: A$
70 PRINT A$
99 END

READY
RUNNH
THIS DATA IS STORED IN A DISK FILE.

READY
```

Note that A$ had to be dimensioned because the input line contained over eight characters. The statement at line 70 simply displays on the screen the data read from the file. Like the PRINT and PRINT# statements, the INPUT and INPUT# statements follow the exact same rules when used with a string file. The only difference is that INPUT reads from the terminal and INPUT# reads from a disk file.

Besides the terminal (reserved as FILE#0), CLASSIC can have up to 4 disk files open at the same time, numbered 1 through 4. On a single disk, however, only one file can be open for writing (as a type V file) at any one time. If you attempt to write to two files on the same disk at the same time, an error message will result.

Exercise 85. To make sure that you can use the file statements discussed so far, write a program similar to the one on the previous page that writes a file containing the first four lines of Lewis Carroll's "Jabberwocky":

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

Verify that your program has run properly by displaying the file on the screen with the monitor TYPE command. Then write a second program that uses the INPUT# statement to read the file and the PRINT statement to display its contents.

Detecting the end of a file. When data is read from a disk file, the system steps through the file in much the same way the READ statement steps through the values in DATA statements. You have seen that the message:

DA AT LINE n

occurs when the end of a data table is encountered. The DA error message automatically causes the program to stop.

In the following example, a second line of data and a GOTO statement have been added to the file program to make it try to read past the end of the file. Look what happens:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 PRINT #1:  "THIS DATA IS STORED IN A DISK FILE."
25 PRINT #1: "THIS IS THE SECOND LINE OF THE DATA."
30 CLOSE #1
40 DIM A$(72)
50 FILE #1: "RXA1:OUTPUT.JH"
60 INPUT #1: A$
70 PRINT A$
80 GOTO 60
99 END

READY
RUNNH
THIS DATA IS STORED IN A DISK FILE.
THIS IS THE SECOND LINE OF THE DATA.
THIS IS THE SECOND LINE OF THE DATA.

RE AT LINE 00060
THIS IS THE SECOND LINE OF THE DATA.<-THIS IS THE CONTENTS
                                      OF A$.
RE AT LINE 00060
THIS IS THE SECOND LINE OF THE DATA.

RE AT LINE 00060
THIS IS THE SECOND LINE OF THE DATA.

RE AT LINE 00060
THIS IS THE SECOND LINE OF THE DATA.

RE AT LINE 00060
THIS IS THE SECOND LINE OF THE DATA.

RE AT LINE 00060
THIS IS TH^C
READY
```

The RE error message indicates that the program tried to read past the end of the data file, but execution is **not** terminated. Instead, the program simply informs the user of the error and continues to print the contents of A$.

CLASSIC provides a special variation of the IF statement to allow you to catch this error. This statement has the form:

line number IF END # file number THEN line number

For example,

65 IF END #1 THEN 90

By adding this statement (and statement 90) to the program on the previous page, you can catch the end of file condition and avoid the error message:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 PRINT #1:  "THIS DATA IS STORED IN A DISK FILE."
25 PRINT #1: "THIS IS THE SECOND LINE OF THE DATA."
30 CLOSE #1
40 DIM A$(72)
50 FILE #1: "RXA1:OUTPUT.JH"
60 INPUT #1: A$
65 IF END #1 THEN 90
70 PRINT A$
80 GOTO 60
90 PRINT "END OF FILE REACHED."
99 END

READY
RUNNH
THIS DATA IS STORED IN A DISK FILE.
THIS IS THE SECOND LINE OF THE DATA.
END OF FILE REACHED.

READY
```

Note that the IF END# statement is placed immediately after the INPUT# statement. This placement is necessary for the statement to work correctly.

Another feature of data files that resembles data tables is the use of the RESTORE# statement. This statement has the form:

line number RESTORE# file number

and causes the pointer indicating the next value to be read to be set back to the beginning of the file. This statement is similar to the RESTORE statement, but it resets a data file rather than a data table.

By adding a RESTORE# statement to the above program as statement number 75, neither the second line of data nor the end of file condition is ever reached:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 PRINT #1:  "THIS DATA IS STORED IN A DISK FILE."
25 PRINT #1: "THIS IS THE SECOND LINE OF THE DATA."
30 CLOSE #1
40 DIM A$(72)
50 FILE #1: "RXA1:OUTPUT.JH"
60 INPUT #1: A$
65 IF END #1 THEN 90
70 PRINT A$
75 RESTORE #1
80 GOTO 60
90 PRINT "END OF FILE REACHED."
99 END

READY
RUNNH
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STORED IN A DISK FILE.
THIS DATA IS STOR^C
READY
```

Exercise 86. Modify the above program so that it displays both lines of data stored in the file before it repeats itself.

Uses of string data files. In Appendix A you will find three programs that make use of string data files for three very different purposes. MORGAG (page A-10) allows the user to direct output to a disk file to speed up processing. The program output may then be displayed on the screen with the monitor TYPE command. ATTEND (page A-3) uses a disk file to store data on student attendance. This file is set up by the program ATTSET and is then *updated* (modified by reading and rewriting the data) by the program ATTEND. The third program, CALC (page A-5), writes an actual BASIC language program in a string data file and then chains to that program and executes it. When finished, the second program chains back to CALC.

Exercise 87. Study the write-ups and listings for the programs mentioned above. Run these programs from the BASIC Program Demonstration Disk and study their output. Display the data files written by these programs on your screen or copier by using the monitor TYPE command. If you wish, add the TRC function to these programs to help you trace their flow. (Do not SAVE any modified versions of these programs on the Demonstration Disk). After you have studied these programs carefully, write a program of your own that both writes and reads a string data file. Try to use all of the file statements introduced so far:

CLOSE#	closes a data file
FILE#	opens a string file so that data can be read from it
FILEV#	opens a string file so that data can be written to it
IF END#	detects the end of an alphanumeric file
INPUT#	reads data from a file
PRINT#	writes data to a file
RESTORE#	resets the file data pointer to the beginning of the file.

Another use of files is to pass data from one program to another during a chain. When the CHAIN statement is executed, all data in the computer's memory is lost. If you wish to use some of it in a chained program, you must first write it into a disk file before chaining and then read it back in after chaining:

```
OLD RXA1:FILE12.BA

READY
LISTNH
100 REM   ***  PROGRAM "FILE12"
110 REM
120 PRINT
130 PRINT "THIS PROGRAM WILL FIND THE STANDARD DEVIATION"
140 PRINT "OF ANY SET OF NUMBERS THAT YOU ENTER."
150 PRINT
160 PRINT "ENTER YOUR SET OF NUMBERS BELOW, AND INDICATE"
170 PRINT "THE END OF YOUR SET BY ENTERING ""-99999""."
180 PRINT
190 PRINT "YOUR NUMBERS";
200 REM
210 REM  ***  DATA INPUT
220 REM
230 DIM N(1000)
240 FOR K=1 TO 1000
250 INPUT H
260 LET N(K)=H
270 IF N(K)=-99999 THEN 320
280 NEXT K
290 REM
300 REM  ***  CREATION OF DATA FILE
310 REM
320 LET K=K-1
330 FILEVN #1: "RXA1:DATAPA.SS"
340 PRINT #1: K
350 FOR KO=1 TO K
360 PRINT #1: N(KO)
370 NEXT KO
380 CLOSE #1
390 CHAIN "RXA1:FILE13.BA"
400 END

READY


OLD RXA1:FILE13.BA

READY
LISTNH
100 REM  ***  PROGRAM "FILE13"
110 REM
120 LET T=0
130 LET T2=0
140 FILEN #1: "RXA1:DATAPA.SS"
150 INPUT #1: A
160 REM
170 REM  ***   DATA INPUT
180 REM
190 FOR K=1 TO A
```

continued on next page

continued from last page

```
200 INPUT #1: H
210 LET T=T+H
220 LET T2=T2+H^2
230 NEXT K
240 REM
250 REM  ***  OUTPUT
260 REM
270 LET S=SQR((T2-(T^2)/A)/A)
280 PRINT
290 PRINT "THE STANDARD DEVIATION OF YOUR"; A; "NUMBERS IS:"
300 PRINT "      "; S
310 REM
320 REM  ***  QUERY FOR ANOTHER RUN
330 REM
340 PRINT
350 PRINT "DO YOU WISH TO ENTER ANOTHER SET OF NUMBERS";
360 INPUT A$
370 IF A$="YES" THEN 430
380 IF A$="Y" THEN 430
390 IF A$="NO" THEN 440
400 IF A$="N" THEN 440
410 PRINT "PLEASE ENTER ONLY ""YES"" OR ""NO""."
420 GOTO 340
430 CHAIN "RXA1:FILE12.BA"
440 END

READY
OLD RXA1:FILE12.BA

READY
RUNNH

THIS PROGRAM WILL FIND THE STANDARD DEVIATION
OF ANY SET OF NUMBERS THAT YOU ENTER.

ENTER YOUR SET OF NUMBERS BELOW, AND INDICATE
THE END OF YOUR SET BY ENTERING "-99999".

YOUR NUMBERS?10,20,30,40,50,60,70,80,90,100,-99999

THE STANDARD DEVIATION OF YOUR 10 NUMBERS IS:
      28.7228

DO YOU WISH TO ENTER ANOTHER SET OF NUMBERS?YES

THIS PROGRAM WILL FIND THE STANDARD DEVIATION
OF ANY SET OF NUMBERS THAT YOU ENTER.

ENTER YOUR SET OF NUMBERS BELOW, AND INDICATE
THE END OF YOUR SET BY ENTERING "-99999".

YOUR NUMBERS?75,80,85,90,95,100,-99999

THE STANDARD DEVIATION OF YOUR 6 NUMBERS IS:
      8.53912

DO YOU WISH TO ENTER ANOTHER SET OF NUMBERS?NO

READY
```

NUMERIC DATA FILES

If you try to read numbers from a string file, problems can occur:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 FOR K=1 TO 10
30 PRINT #1: K
40 NEXT K
50 CLOSE #1
60 FILE #1: "RXA1:OUTPUT.JH"
70 FOR K=1 TO 10
80 INPUT #1:N
90 PRINT N
95 NEXT K
99 END

READY
RUNNH
 1
 0
 0
 2
 0
 0
 3
 0
 0
 4

READY
```

The computer seems to have written two zeroes in between each of the data values. This is not really the case. The real reason for this result is explained below.

You will remember that after a PRINT statement is executed, the cursor is always positioned at the beginning of the next line unless the PRINT statement ends with a comma or a semicolon. CLASSIC does this by sending two special characters to the terminal. These are called the **carriage return** and **line feed** characters and are exactly the same as those sent by the PNT(13) and PNT(10) functions, respectively. The carriage return and line feed characters are also written when CLASSIC outputs data to a string disk file. When these characters are read back into numeric variables by the INPUT# statement, they are interpreted as zeroes.

To help you understand this, look at the following diagram. This diagram shows the actual characters that are stored in the alphanumeric file written by statements 10 to 50 of the above program. (Each box represents one character position).

SPACE	1	SPACE	carriage return	line feed	
SPACE	2	SPACE	carriage return	line feed	
SPACE	3	SPACE	carriage return	line feed	
SPACE	4	SPACE	carriage return	line feed	
SPACE	5	SPACE	carriage return	line feed	
SPACE	6	SPACE	carriage return	line feed	
SPACE	7	SPACE	carriage return	line feed	
SPACE	8	SPACE	carriage return	line feed	
SPACE	9	SPACE	carriage return	line feed	
SPACE	1	0	space	carriage return	line feed

When this file is read back with the FOR-NEXT loop at lines 70-95, the spaces are ignored but the carriage return and line feed characters are interpreted as zeroes. Thus, the first ten numbers interpreted by the INPUT# statement are as follows:

1 0 0 2 0 0 3 0 0 4

One way to correct this problem is to always read data from string files as strings and then convert numbers to numeric data by using the VAL function:

```
10 FILEV #1:  "RXA1:OUTPUT.JH"
20 FOR K=1 TO 10
30 PRINT #1: K
40 NEXT K
50 CLOSE #1
60 FILE #1: "RXA1:OUTPUT.JH"
70 FOR K=1 TO 10
80 INPUT #1: N$
90 PRINT VAL(N$)
95 NEXT K
99 END

READY
RUNNH
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10

READY
```

A better method is to restrict the use of string files to string data and to use **numeric** type files to store numeric data. Files can be opened as numeric type by adding the letter N to the FILEV and FILE statements:

10 FILEVN #1: "RXA1:OUTPUT.JH"
60 FILEN #1: "RXA1:OUTPUT.JH"

When numeric files are used, the problem caused by the carriage return and line feed characters disappears:

```
10 FILEVN #1: "RXA1:OUTPUT.JH"
20 FOR K=1 TO 10
30 PRINT #1: K
40 NEXT K
50 CLOSE #1
60 FILEN #1: "RXA1:OUTPUT.JH"
70 FOR K=1 TO 10
80 INPUT #1: N
90 PRINT N
95 NEXT K
99 END

READY
RUNNH
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10

READY
```

No special conversion needs to be performed; the data values can be read directly into numeric variables with successive INPUT# statements. Writing a numeric data file is similar to creating a numeric data table: no matter how many PRINT# statements are needed to write the file, all the data will be considered as a single set, and all spaces, carriage returns, and line feeds will be filtered out.

There are two other differences between numeric files and string files. First, you will remember that it was possible to write numbers into a string file. It is not possible, however, to write strings into a numeric file:

```
10 FILEVN #1:  "RXA1:OUTPUT.JH"
20 FOR K=1 TO 10
30 PRINT #1: "NUMBER"; K
40 NEXT K
50 CLOSE #1
60 FILEN #1: "RXA1:OUTPUT.JH"
70 FOR K=1 TO 10
80 INPUT #1: N
90 PRINT N
95 NEXT K
99 END

READY
RUNNH

SW AT LINE 00030

READY
```

The SW error message indicates that an attempt was made to write a string into a numeric file and causes termination of the program.

Second, the end of a numeric file cannot be recognized by the system:

```
10 FILEVN #1: "RXA1:OUTPUT.JH"
20 FOR K=1 TO 10
30 PRINT #1: K
40 NEXT K
50 CLOSE #1
60 FILEN #1: "RXA1:OUTPUT.JH"
80 INPUT #1: N
90 PRINT N
95 GOTO 80
99 END

READY
RUNNH
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 0
 0
 0
 0
 0
 0
 0
 0
 0
^C
READY
```

If CTRL/C had not been typed during the above run, the system would have gone on printing zeroes indefinitely. The inability of the system to recognize the end of a numeric file also means that the IF END# statement cannot be used with these files.

Exercise 88. Write a program of your own that makes use of a numeric data file and includes the FILEVN# and FILEN# statements. Try different arrangements of the expressions in a PRINT# statement to see if they have any effect on the program's ability to reread the data. For example, find out if the statement:

30 PRINT #1: K, K+1, K+2

causes the same result as:

30 PRINT #1:K; K+1; K+2

and:

30 PRINT #1: K
33 PRINT #1: K+1
35 PRINT 1: K+2

Devise further experiments of your own to test the ways in which numeric files differ from string files.

Exercise 89. When data tables become large, they can occupy so much of the computer's memory that little is left for the program itself. Therefore, large amounts of data are often stored in files rather than data tables. Modify the TALLY program on page 3-39 so that it reads data from a disk file rather than a data table. Write a separate program to create the file from the original data table.

LOOKING BACK

You have now been introduced to all of the BASIC statements that are available on your CLASSIC system. Chapter 4 of the *CLASSIC User's Reference Guide* summarizes these statements and provides an alphabetical reference for you to use while you are programming.

The programs in Appendix A demonstrate the many different types of things that you can program CLASSIC to do using the BASIC language. Chapter 5 discusses these programs fully and suggests some programming guidelines that you might follow to make your programs easier for others to use.

The final section in this chapter will help you learn how to use the remaining monitor commands.

SECTION 4-E
USING MONITOR COMMANDS

In Chapter 1 you learned the monitor command R BASIC, and in Section 3-E you learned the monitor commands DATE, DELETE, DIRECT and TYPE. This section will introduce you to the other eight CLASSIC monitor commands and show you new ways to use the commands that you already know.

VARIATIONS OF THE DIRECT COMMAND

If you are looking for a specific file on a disk, you need not list the entire directory. Just type in the name of the file that you are looking for and CLASSIC will display its directory entry if it is present:

```
.DIRECT RXA1:CALC.BA
```

```
CALC   .BA    4
```

If it is not present, only the number of free blocks on the disk will be displayed.

Up to nine different file names may be entered on one line. The example below shows the abbreviation for the DIRECT command and requests that the directory entries for two files be displayed if they are present:

```
.DIR RXA1:SYNONY.BA,SYNSET.BA
```

```
SYNONY.BA   12
SYNSET.BA    1
```

Note that the device name for SYNSET.BA is not specified. CLASSIC assumes that SYNSET.BA is on RXA1 because the preceding file is on RXA1. In this command, each file specified is considered as an *input* entry to the DIRECT command (see page 3-20).

In any list of input entries, each file is assumed to be on the same device as the preceding file.

Entering options. Some monitor commands allow you to enter *options* which affect the functioning of the command. Options can be either letters or numbers. If they are letters, options are preceded by slashes (/).

One option recognized by the monitor DIRECT command is /F. This option causes a "fast" directory to be printed, displaying only the file names:

```
.DIRECT RXA1:/F
```

```
ACEY02.BA
ATTEND.BA
ATTSET.BA
CALC  .BA
EASY02.BA
EASY03.BA
GUESS .BA
HMRABI.BA
HURKLE.BA
HURK02.BA
MORGAG.BA
QUADEQ.BA
QUAD02.BA
QUAD03.BA
SYNONY.BA
SYNSET.BA
WTDAVG.BA
```

When a number option is used, it always follows an equal sign (=). When used with the DIRECT command, the number option indicates the number of columns to be used in displaying the output:

```
ACEY02.BA  24          ATTEND.BA  24
ATTSET.BA   2          CALC  .BA   4
EASY02.BA   1          EASY03.BA   4
GUESS .BA   5          HMRABI.BA  22
HURKLE.BA   4          HURK02.BA  31
MORGAG.BA  10          QUADEQ.BA   2
QUAD02.BA   4          QUAD03.BA   5
SYNONY.BA  12          SYNSET.BA   1
WTDAVG.BA  19
```

More than one option may be entered in a single command line:

```
.DIR RXA1:/F=5
```

```
ACEY02.BA  ATTEND.BA  ATTSET.BA  CALC  .BA  EASY02.BA
EASY03.BA  GUESS .BA  HMRABI.BA  HURKLE.BA  HURK02.BA
MORGAG.BA  QUADEQ.BA  QUAD02.BA  QUAD03.BA  SYNONY.BA
SYNSET.BA  WTDAVG.BA
```

Exercise 90. Insert the System disk in drive 0 and the BASIC Program Demonstration disk in drive 1. List the directories of both disks using the options discussed above.

The wild card construction. Wild cards are used to replace all or part of the file name or extension in a monitor command line.

A wild card may be either an asterisk or a question mark. An asterisk replaces an entire file name or extension while a question mark replaces only a single character.

To list only those files on the system disk with the extension .SV, enter the following command:

```
.DIR RXA0:*.SV

CCL    .SV  17 21-JAN-75
DIRECT.SV    7 08-MAY-75
FOTP   .SV   8 08-MAY-75
PIP    .SV  11 08-MAY-75
BCOMP  .SV  17 18-JAN-74
BASIC  .SV   9 13-MAY-75
BLOAD  .SV   7 13-MAY-75
BRTS   .SV  15 08-MAY-75
```

The above command uses the asterisk wild card to replace the file name in the input entry. When the computer searches the disk directory for files that match the input entry, any file with an extension of .SV is accepted because all file names match the asterisk wild card.

The asterisk wild card can also be used to replace the file extension:

```
.DIR RXA0:BASIC.*

BASIC .AF   4 18-JAN-74
BASIC .SF   4 08-MAY-75
BASIC .FF   4 18-JAN-74
BASIC .SV   9 13-MAY-75
BASIC .UF   4 18-JAN-74
BASIC .WS   6 20-JUL-75
```

In this case, the directory entries for all files with the name BASIC and any extension are displayed.

The question mark replaces individual characters in the file name or extension. When searching for files to match the input entry, the computer accepts any characters in the positions containing question marks. The following command therefore displays the directory entries for all files whose names start with B and have an .SV extension:

```
.DIR RXA0:B?????.SV

BCOMP .SV  17 18-JAN-74
BASIC .SV   9 13-MAY-75
BLOAD .SV   7 13-MAY-75
BRTS  .SV  15 08-MAY-75


 153 FREE BLOCKS
```

The next command displays all files whose names start with B and have an .SV extension but whose names are not more than four characters long:

```
.DIR B???.SV


BRTS  .SV  15 01-JAN-75
```

The question mark wild card may not be used in an output file entry.

Options may be used in a command line containing wild cards:

```
.DIR RXA0:B?????.*/F=5

BCOMP .SV     BASIC .AF     BASIC .SF     BASIC .FF     BASIC .SV
BLOAD .SV     BRTS  .SV     BASIC .UF     BLKJAK.BA     BLKJAC.BA
BASIC .WS
```

Exercise 91. More examples of wild cards and options used with the monitor DIRECT command are presented in the *CLASSIC User's Reference Guide.* Study these examples and experiment with wild cards by displaying various directory listings of the system and demonstration disks on your screen. For example, try entering the following commands to see what happens:

```
.DIR RXA0:B*.SV
.DIRECT SYS:*.*
.DIR DSK:
.DIR RXA1:??*.??
.DIR *.?F
.DIRECT .SV
.DIR SYS:BASIC
.DIR RXA1:???Y??.BA
```

RENAMING DISK FILES

So far, you know how to create disk files by writing BASIC language programs or outputting program data to a disk. You also know how to erase files with the monitor DELETE command. If you wanted to change the name of a program file, you could therefore do it by the following procedure:

(1) Start up the BASIC editor and read the file to be renamed into the workspace with the editor OLD command.
(2) SAVE the file under a new name.
(3) Return to the monitor and erase the old copy of the file with the DELETE command.

Not only is the above procedure rather clumsy, but it does not work for data files. CLASSIC therefore provides the RENAME command to simplify this task.

The monitor RENAME command changes the names of disk files.

The general format of the RENAME command is as follows:

```
.RENAME dev:newfile.ex<dev:oldfil.ex
```

(The command word RENAME may be abbreviated to REN.) The device entry on both sides of the < must be the same.

Exercise 92. Insert the system disk into drive unit 0 and a "scratch" disk (one that has files on it but that you can afford to erase) into drive unit 1. List the directory of the scratch disk, and then change the name of one of its files using the monitor RENAME command. Verify that the file has been renamed by listing the disk directory again.

The RENAME command will also accept wild cards. With any monitor command besides DIRECT, it is recommended that you always include the /Q option. This option will query (ask) you about each file to be affected. A response of Y will cause the indicated operation to take place, while a response of N will cause that file to be skipped. The /Q option works with wild cards like this:

```
.RENAME RXA1:*.RN<RXA1:*.TM/Q
FILES RENAMED:
ONE. TM?Y
TWO. TM?N
THREE. TM?Y
FOUR. TM?N
FIVE. TM?Y

.DIR RXA1:*.TM,*.RN


ONE     .RN    1 30-AUG-76
TWO     .TM    1 30-AUG-76
THREE   .RN    1 30-AUG-76
FOUR    .TM    1 30-AUG-76
FIVE    .RN    1 30-AUG-76
```

Note that the asterisk on the left side of the < in the above command line has a different significance from the one on the right. In an input file entry (on the right), the wild card means any file name or extension. In an output entry (on the left), the wild card means "use the same name or extension as in the input entry". Therefore, only the file extensions will be changed by the above command.

Exercise 93. Experiment with the RENAME command using wild cards and the /Q option to change the names of other files on your scratch disk.

Do NOT rename the files on RXA0 or your system will not operate properly.

COPYING DISK FILES

The COPY command. Very often, you will find it useful to copy a disk file from one disk to another. This can be done with the monitor COPY command. This command has the form:

.COPY dev:output.ex<dev:input.ex

After this command is executed, the output file will be an exact copy of the input file. For example,

.COPY SYS:LOAN.DM<RXA1:MORGAG.BA

will cause a copy of MORGAG.BA on RXA1 to be created on SYS (RXA0). The new copy will be named LOAN.DM.

If the name of the file is not to be changed, only the output device needs to be specified in the output entry. Each of the following three commands will therefore accomplish the same task:

.COPY RXA0:MORGAG.BA<RXA1:MORGAG.BA
.COPY RXA0:.* < RXA1:MORGAG.BA*
.COPY RXA0:<RXA1:MORGAG.BA

Like the DIRECT command, the COPY command accepts wild cards and different options. For example, the following command line will copy all files with .BA extensions from RXA0 to RXA1, querying the user before each copy is made:

.COPY RXA1:<RXA0:.BA/Q*

Exercise 94. The complete set of options allowed by the COPY command and examples of their use are presented in the *User's Reference Guide.* Read these pages and experiment with the COPY command by transferring files from RXA0 to RXA1 and back again. Once again, do **not** change the name of the original system files on RXA0 or your system will not function properly.

The ZERO command. Sometimes it is desirable to delete all the files on a disk. This can be done in two ways. One way is to use the DELETE command with wild cards for both the file name and extension entries. With this method, it is possible to accidentally erase all the files on your system disk and therefore destroy the CLASSIC software.

A safer method is to use the ZERO command. This command tells CLASSIC to erase all files on the disk specified. If you try to zero the System disk by mistake, CLASSIC will respond:

ZERO SYS?

Always respond N to this query to avoid losing the system software.

The monitor ZERO command should thus be used only in the following form:

.ZERO RXA1:

Exercise 95. Try out the monitor ZERO command by following these steps:

(1) Copy all of the files on your scratch disk onto the system disk by entering:

.COPY SYS:< RXA1:

(2) List the directory of RXA1: to verify that its files are still present.

(3) Zero your scratch disk by entering:

.ZERO RXA1:

(4) List the directory of RXA1: to verify that its files have been erased.

The SQUISH command. There are two ways to put the files back onto RXA1 by copying them from RXA0. One way is to use the monitor COPY command in the form:

.COPY RXA1:<SYS:

A faster way is to use the SQUISH command:

.SQUISH RXA1:<SYS:

The SQUISH command has two advantages and two disadvantages when compared with the COPY command. Its advantages are that it is faster than the COPY command and that it automatically eliminates any gaps between files on the output disk. Its disadvantages are that it automatically zeroes the output disk (erasing any previously stored files) and cannot be interrupted by a CTRL/C. To counteract these disadvantages CLASSIC therefore prints the message:

ARE YOU SURE?

before a SQUISH command is executed. A response of Y will cause the SQUISH to occur; N will return control to the monitor without execution of the command.

Exercise 96. The SQUISH command is very useful when you want to make an exact copy of the RXA0 disk. Enter the following command to your system:

.SQUISH RXA1:<SYS:

When CLASSIC prints ARE YOU SURE?, check your command line to be sure that it is typed correctly and then respond with Y.

WARNING: Never specify SYS:, DSK:, or RXA0: as the output device in a SQUISH command. This action will destroy the CLASSIC system software.

If you now compare the directories of RXA1 and RXA0, you will see that they are exactly the same. Try the following experiment. Take the System Disk out of RXA0 and place it on the desk. Move the other disk from RXA1 to RXA0. Then try to restart the system.

What happened? Probably nothing. The system would not work because the disk in the drive does not contain the monitor program.

The monitor program is the only file ever stored on a disk that does not have an entry in the disk directory.

To copy the monitor program, you must use the R PIP command.

The R PIP command. To copy the monitor program, it is necessary to type the following lines:

.R PIP

**RXA1:< (YZ)* Push ESC Key

This is a two-line command and has no variables. It should be typed exactly as shown above. When you push the ESC key, the system will display a dollar sign ($).

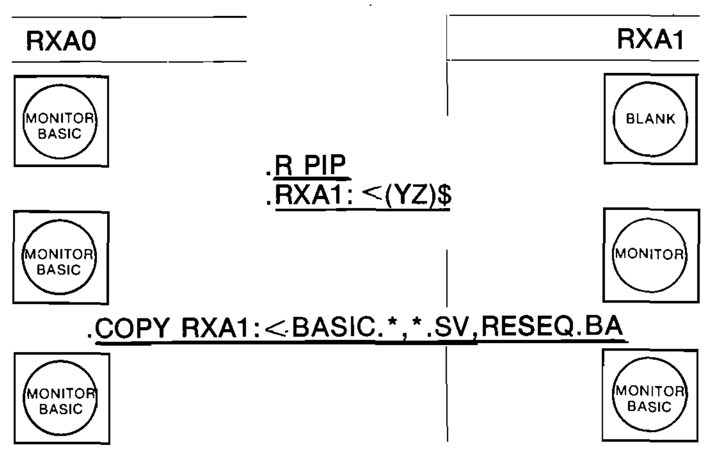
This command erases all files from RXA1 and then writes the monitor program on that disk. When it is completed, the monitor dot should reappear. If an asterisk (*) appears rather than the dot, type CTRL/C to return to the monitor.

Once you have copied the monitor program, you can copy all the other files needed to create a CLASSIC System disk by typing:

.COPY RXA1:<BASIC.,*.SV,RESEQ.BA*

Exercise 97. Enter the above commands to your system and then verify that the new disk can be used as a system disk by inserting it in RXA0, starting the system, and running the BASIC editor.

The procedure outlined above (which creates a new copy of the CLASSIC System Disk) can be clarified by the following diagram:



STORING PROGRAMS IN COMPILED FORM

Exercise 98. Obtain a watch with a second hand, and then type the following commands:

.R BASIC
NEW OR OLD—OLD RXA1:HURK02
READY

now type RUNNH but do **not** press the RETURN key until you note the position of your watch's second hand. Determine how long it takes for CLASSIC to begin executing HURK02 from the time that you press the RETURN key. Note the elapsed time on a separate piece of paper.

As soon as you press the return key, CLASSIC begins to **compile** your program. This means that the program is translated into an internal form that can actually be executed by the computer. Compilation usually takes only a few seconds, but with a very large program, like HURK02, it can take a little longer. When programs are no longer going to be changed, it is often convenient to store them in their compiled form so that the program does not have to be recompiled each time that it is run.

To store a program in compiled form, you must use two monitor commands. The first command causes the BASIC language program to be compiled and the compiled form to be placed in the computer's memory. This is a two-line command of the following form:

.R BCOMP
**dev:filnam.ex/K=3*

The dev:filnam.ex entered as a parameter to this command should specify the BASIC language program that you wish to compile. For example,

.R BCOMP
**RXA1:HURK02.BA/K=3*

The /K and =3 **must** be included in this command in order for it to work properly. /K tells the computer to translate the indicated program into compiled form, and =3 tells it the amount of memory that is available for use.

To store the compiled form of your BASIC language program on a disk, type:

.SAVE dev:filnam

where dev: is the device name on which you want the compiled program stored, and
filnam is the file name that should be used in storing the program.

No file extension should be specified; CLASSIC automatically appends the extension .SV to programs stored in compiled form. For example,

.SAVE RXA1:HURK02

There will now be two programs named HURK02 on RXA1, HURK02.BA and HURK02.SV. Note the difference between this command and the editor SAVE command:

The editor SAVE command stores programs in their BASIC language form. The monitor SAVE command stores programs in their compiled form.

To run a compiled program, you must use the monitor RUN command. For example,

.RUN RXA1:HURK02

This command has the same general format as the monitor SAVE command:

.RUN dev:filnam

CLASSIC automatically looks for a file with the extension SV.

The editor RUN command executes BASIC language programs. The monitor RUN command executes programs stored in compiled form.

The complete process of compiling, storing, and running a BASIC language program is shown below:

```
.R BCOMP
*RXA1:HURK02.BA/K=3

.SAVE RXA1:HURK02

.RUN RXA1:HURK02
 HURKLE TWO
----------

DO YOU WISH TO SEE THE INSTRUCTIONS ('YES' OR 'NO')?
```

Exercise 99. Enter these commands into your computer and record how long it takes for CLASSIC to begin executing the program once you press the RETURN key after the RUN command. Compare this time to the value that you found when using the editor RUN command.

There is one special rule that you must follow when storing and running programs in compiled form:

BASIC language programs can only chain to other BASIC language programs, and compiled programs can only chain to other compiled programs.

Exercise 100. Experiment with this rule by storing and running the TARGET and CHAIN programs that were demonstrated on page 4-28 in both BASIC language and compiled form. When both programs are in compiled form, the CHAIN statement must be in the form:

30 CHAIN "RXA1:TARGET.SV"

Note the difference in the time required to chain between programs in BASIC language and compiled forms.

LOOKING BACK

This section concludes the introduction to all of the commands and statements that make up the CLASSIC software. Additional examples and notes on the monitor commands discussed in this section can be found in Chapter 2 of the *User's Reference Guide*. The commands in that chapter are described in alphabetical order.

Throughout Chapter 4, you have seen many examples of BASIC programs which apply CLASSIC's capabilities to a variety of tasks. In Chapter 5, you will be introduced to more sophisticated CLASSIC applications. By studying Chapter 5 and the accompanying programs in Appendix A, you will learn additional tricks of the programming trade and get some new ideas about programs that you can write for CLASSIC.

Chapter 5

Classic Applications

UNDERSTANDING WHAT TO DO

This chapter introduces you to some of the things that you should consider when you apply CLASSIC's capabilities to certain tasks.

The chapter is broken down into five lessons or **modules.** The first module discusses the types of computer applications that are found in today's schools and colleges and presents examples of programs that can be run on your CLASSIC system. The second, third, and fourth modules will help you learn about:

(1) how to plan a large computer program,
(2) ways to make it easier for others to use your programs, and
(3) some things to consider so that your programs can be used on computers other than CLASSIC.

The final module lists books and magazines that you may use to teach yourself more about computers in education.

Each module contains the following five sections:

(1) What You Will Do
(2) How Far You Should Go
(3) Things You Will Need
(4) What It's All About
(5) Self-Test

Each of these sections is described below to guide you in using them effectively.

What you will do. The first part of each module is an exact statement of what you will be able to do when you have completed the module. All the information, examples, and learning activities presented in the module are designed to help you do what is stated. You should read this statement carefully to understand the purpose of each module and avoid unnecessary work.

How far you should go. This section describes how much you should work on the module before you proceed to the next one. If you think that you already know the material to be covered as it is described here, test yourself with the self-test.

Things you will need. The section is a list of all the materials you will need to do the learning activities in the module. For example, some modules require CLASSIC while others do not, and some require a special demonstration disk. This section will help you plan your work.

What it's all about. This is the largest section of each module. It discusses (1) the information needed to complete the module, (2) examples of CLASSIC usage, and (3) learning activities that you can carry out.

Self-test. Each module ends with a self-test that you can take to measure your learning. This test will let you know whether you are ready to go on to the next module or whether you need more practice on the current one. If you think that you can pass this self-test without going through the "What It's All About" section, go ahead and try. If you do pass the test, go on to the next module. If you do not, work more carefully on the activities in "What It's All About".

DISKS

There are two disks which are referred to in this chapter and will be needed to carry out the learning activities. The first is called the CLASSIC System disk and contains the CLASSIC system software. The second is the BASIC Program Demonstration disk and contains sample programs for you to run. Copies of these disks should be available from your instructor or the person in charge of your CLASSIC (your system manager).

MAKE SURE THAT YOUR SYSTEM MANAGER HAS BACK-UP COPIES OF BOTH DISKS BEFORE YOU WORK ON THIS CHAPTER.

Instructions for making back-up copies of your disks may be found in Chapter 1 of the *CLASSIC Installation and Maintenance Guide.*

MODULE 5-A

EXAMPLES OF CLASSIC APPLICATIONS

WHAT YOU WILL DO

Examine the programs on the BASIC Program Demonstration disk that demonstrate instructional computer applications. Look for situations in your own school or community in which programs of these types might be useful. Then choose one of the program descriptions on page 5-4 to use as a basis for developing your own application program in later modules.

HOW FAR YOU SHOULD GO

For each of the following four categories of instructional computer applications, describe at least one situation in your own school or community in which CLASSIC might be applied.

(1) Administration
(2) Computer-Assisted Instruction (CAI)
(3) Problem Solving
(4) Simulation

THINGS YOU WILL NEED

(1) CLASSIC
(2) CLASSIC System disk
(3) BASIC Program Demonstration disk

WHAT IT'S ALL ABOUT

Figure 5-1 shows some of the most common instructional computer applications. (Many computer programs fall into more than one of these categories.) Each of these applications will be discussed, and the programs on your demonstration disk will be used as examples. Each program has a **write-up** that explains how it works. These write-ups appear in Appendix A.

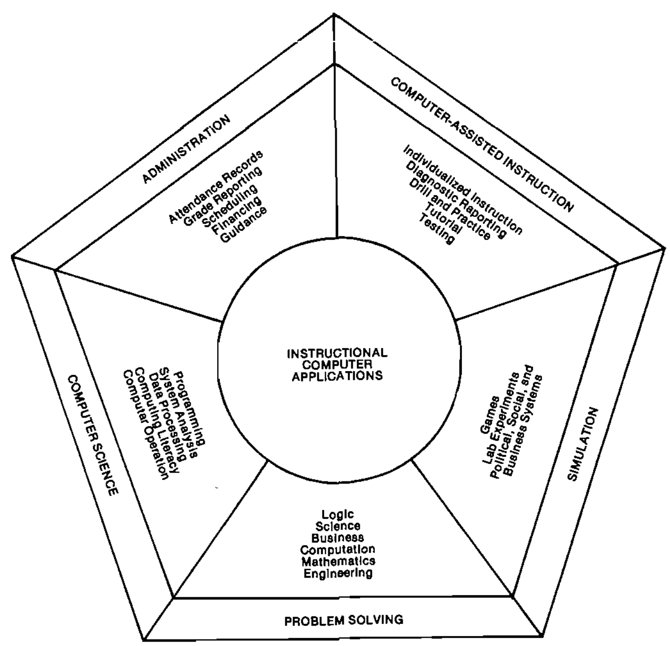


Figure 5-1

Instructional Computer Applications

Administration. The computer has become an important part of almost every large organization in the world. In fact, the administrative jobs that computers do are so sizable that most of our large schools and businesses would find it difficult to do their jobs without computers.

In schools and colleges, computers are used to keep student records, improve class scheduling procedures, keep financial records, print grade reports, and supply guidance information.

CLASSIC is a small computer compared to those that may handle the administration of your school and community, but it can still be used to do some types of administrative tasks. One of these tasks might be weighted grade averaging, a simple calculation that applies a "weighting" effect to the averaging of a set of numbers. This task is often used by teachers who wish to "average" quizzes, papers, and tests to arrive at a final numerical grade but who feel that short quizzes should count less than large exams. The program that demonstrates this task is called WTDAVG ("Weighted Averaging"). Read the write-up for this program on page A-13 and run the program from the demonstration disk.

The second task involves file storage — saving information from one run of a program to be used in another. Program ATTEND demonstrates this task by keeping a record of student attendance. This program is also available on your demonstration disk and its write-up is on page A-3. Try it before you go on.

Computer-Assisted Instruction. CAI is a term that has been applied to many different instructional computer applications. In its most general sense, CAI includes **any** use of the computer to assist instruction. More often, the term CAI refers to applications in which students run programs written by an instructor and interact with the computer system by answering questions printed on the terminal. This limited application is also called drill-and-practice or tutorial CAI. CAI also includes the use of the computer to test students. In this application, the computer usually stores information on each student's score and can display this data for the instructor.

The program on your demonstration disk that illustrates CAI in drill-and-practice and testing is called SYNONY, and its write-up appears on page A-12. This program uses the disk to store questions, answers, and information about student performance. Try it out before you go on.

Computer Science. The amazing growth in the number of organizations that use computers has created many jobs for people with computer experience. Computer science is a wide field that includes skills such as programming and general computer operations, topics that your CLASSIC is well suited to helping you learn. (Chapter 3 is designed to be an introduction to this field.)

Computer literacy is also a part of computer science. A person who is computer literate is one who has some idea of how computers operate, how people use them, and what their capabilities and limitations are. (This chapter introduces you to some concepts in computer literacy.) CLASSIC can be used to demonstrate each of these principles and can provide a stepping stone to advanced courses in computer science.

Problem solving. In this application, the computer is used to do the tedious calculations that are usually done by hand. Problem solving is the oldest application of calculational machines. It was for the purpose of speeding up mathematical computations that the abacus, adding machine, slide rule, calculator, and computer were invented. The application of the computer to problem solving in instructional situations is limited only by imagination.

Several programs are included on the demonstration disk that illustrate problem solving. These are:

CALC	calculates the value of any valid CLASSIC mathematical expression (write-up on page A-5)
EASY02	finds the factors of a given number (page A-5)
MORGAG	computes mortgage payments (page A-10)
QUADEQ	solves quadratic equations (page A-11)

Look at the write-ups for these programs and run them from your demonstration disk.

Simulation. This final category of instructional computer applications is among the most popular uses of the computer in education. This category includes games and simulated experiments, programs that allow the user to match his or her wits against the computer and run experiments without any real risk.

There are three programs on your demonstration disk that have educational value as well as being enjoyable to run. The first of these is HMRABI, which allows you to act as Hamurabi, the governor of the ancient city of Sumeria. This simulation demonstrates the importance of balance in running the affairs of state. The write-up is on page A-6.

The second program is called HURKLE. This is a game which tests your knowledge of the Cartesian coordinate system (see the write-up on page A-7) by finding a "Hurkle" hiding within a grid. With a little bit of practice, you should be able to find the Hurkle with a relatively small number of guesses. HURK02 is a more challenging version of this game (see page A-8).

The last application program is ACEY02 (the write-up is on page A-1). This is a version of the Acey-Deucey card game. Using ACEY02 you can gamble hundreds of dollars without opening your wallet. There are a few tricks of the trade, however, and some thinking about probabilities will greatly increase your winnings.

Try running each of these programs.

SELF-TEST

Now that you have been introduced to some instructional computer applications, look around your school, college, or community for ways to apply CLASSIC. Describe at least one task for each of the four application areas specified in "How Far You Should Go". Discuss the descriptions that you write with people who might use your ideas. These discussions will help you to further define the tasks and understand if your ideas will work.

Listed below are descriptions of programs in four different applications areas. Choose one of these descriptions to use as a basis for developing your own application program in later modules. (If none of these suit you, you may make up your own.)

(1) Administration:

(a) Frequency - write a program that analyzes a set of scores and displays a bar graph showing the number of times that each score was achieved.

(b) Inventory - this program might maintain a file of all the audio-visual equipment in your school or college, including usage data as well as the type, cost, supplier, and purchase date of each machine.

(2) Computer-Assisted Instruction

(a) Fractions - present 10 multiple choice fraction problems in the following form:

$$\frac{2}{3} + \frac{1}{4} =$$

(A) $\frac{3}{7}$ (B) $\frac{3}{12}$ (C) $\frac{11}{12}$ (D) $\frac{6}{4}$

YOUR ANSWER (A,B,C, OR D)?

(b) Spelling - display a sentence on the screen containing a misspelled word. Then ask the user to type the correct spelling of the word in error.

(3) Problem Solving:

(a) Cubic - expand the QUAD03 program on the BASIC Program Demonstration disk to solve cubic equations of the form:

$Ax^3 + Bx^2 + Cx + D = 0$

(b) Bounce - create a program that will diagram the bouncing of a rubber ball.

(4) Simulation:

(a) Games - almost any game can be simulated on the computer. If you have a favorite one, write a program that simulates it.

(b) Titration - simulate an acid-base titration in chemistry. Your program should reflect as many of the important factors in a real titration as possible.

MODULE 5-B

PLANNING PROGRAMS FOR CLASSIC

WHAT YOU WILL DO

Plan and write an application program for the description that you selected at the end of Module 5-A. Your program need not be very long, but it should be sizable enough to use many different types of BASIC statements (perhaps 50 - 100 lines).

HOW FAR YOU SHOULD GO

Discuss your program plan with your instructor or system manager. He or she should agree that your plan is a good one for CLASSIC and that your program is not too large to be completed within a reasonable amount of time. Write your program and save your work on a disk for use in Modules 5-C and 5-D.

THINGS YOU WILL NEED

(1) CLASSIC
(2) CLASSIC System disk
(3) Scratch or blank disk (if available)

WHAT IT'S ALL ABOUT

In this module, you will develop a complete computer program, planning the program as a whole rather than building on pieces of programs as in Chapter 4. By doing this, you will better understand the work involved in writing programs and be better prepared to write programs for any task.

In planning your program, follow the simple stepwise procedure defined below.

(1) Write down your idea.
(2) Consider your time.
(3) Consider CLASSIC's capabilities.
(4) Define exactly what you want to do.
(5) Test your programming techniques on the computer.
(6) Write your program off-line.
(7) Enter your program.
(8) Debug your program.
(9) Document your program.

Write down your idea. The first step in planning any program is to describe what your program will do as carefully as you can. Your description should be precise and complete. For example, the following description is **not** good enough to build a program from:

> The program will calculate how much you must pay back if you borrow a certain amount of money.

This is better:

Given:

(1) the amount of money borrowed,
(2) the yearly interest rate to be paid, and
(3) the number of years allowed to pay back the loan,

the program will calculate:

(1) the monthly interest rate,
(2) the number of months allowed to pay back the loan, and
(3) the amount of money to be paid each month.

Note that both input and output are specified. (This is a description of the first part of MORGAG, the application program written up on page A-10.)

Expand upon the program description that you have chosen by specifying what data you will supply, what the computer will do with it, and what results will be printed. Describe your intended program as carefully as you can.

Consider your time. Consider how long it will take you to write the program that you have described. Your work in Chapters 3 and 4 probably convinced you that computer programming can be a time-consuming task. If you are programming simply for fun and have no special date by which your program must be completed, you need not be too concerned about the time. But if you have a deadline, you might do better to trim your idea down to a more reasonable size. You can usually come back and add your other ideas later.

Consider CLASSIC's Capabilities. A second point that beginners often fail to consider is the capability of their computer. CLASSIC is a powerful machine, but it has limitations just like any other machine. As you plan, keep the following points in mind:

(1) CLASSIC has a finite memory size. The computer can handle very large programs, but only if they do not contain large arrays. (Remember that the RESEQ program can only handle up to 350 lines.) If your programs are too big for CLASSIC, break them up and use the CHAIN command.
(2) Your disks also have finite size, especially if you store all your programs and data files on the system disk. For programs using very large amounts of data, plan to use a blank disk for your files instead of the system disk.
(3) Four files may be open at once, and only one of these may be used for writing.
(4) Print-out is limited to 72 columns. Therefore, any line longer than 72 columns will have its extra characters printed on the next line.
(5) CLASSIC may be used by one student at a time or by a group of students together. Programs designed to collect data from many students one at a time should be kept short. For example, consider what would happen if 30 students each had to interact independently with a program for 10 minutes. The entire process would require 300 minutes, or 5 hours, or almost every minute of an entire school day.

Think about the considerations mentioned above and modify your original written idea, if necessary, before you go on.

Define exactly what you want to do. Take your carefully considered idea and break it down into fine detail. At this point in the planning process you may wish to draw a generalized **flowchart** for your program to show how it will work. Flowcharting was introduced in Chapter 3 to show what the computer does when you give it certain instructions. Generalized program flowcharts are less detailed. Figure 5-2 is a flowchart of the program planning task that is being discussed in this module. You will remember that each block contains instructions on activities to be carried out. Decision points are shown with a diamond, and entry and exit points are shown with an oval. At this point, make a generalized flowchart and discuss your program plan with your instructor or system manager as described in the first part of "How Far You Should Go". Before you begin programming, you will want to make absolutely sure that your time will be well spent.

Test out needed programming techniques on the computer. Your detailed plan will call for certain tasks to be done in certain ways. A programmer is seldom completely sure that his or her approach to a problem is correct until it is actually tried on the computer. The programmer may have a wrong idea about the use of a specific program statement or how the computer executes a specific series of statements. These are called logic errors, as no error messages are printed by the computer but the program does not do what the programmer wishes.

Before you begin writing your complete program, test your logic by writing simple programs to try out your ideas.

Write your program. Don't make the mistake of trying to write your program at the keyboard. This is extremely difficult and time-consuming. Large programs are best written on a piece of paper rather than while sitting at the terminal.

Enter your program to the system. With your program written, you should have little trouble typing it into your system. Be sure that you SAVE your program on a disk so that you will not have to retype it the next time you want to use it. It is a good idea to SAVE your program after every 25 to 30 lines that you type because you may mistakenly delete lines or your entire program and lose a good deal of work. If you are working on a very large program, ask your system manager if you can borrow a disk for your own use.

Debug your program. Programming errors are known as **bugs,** and the process of correcting them is known as **debugging.** It is unusual to enter a program, type RUN, and not receive any error messages. Programmers usually make at least one typing error in every five to ten lines. These errors are easy to correct, because the computer prints out each error that it finds and the line numbers in which the errors occur. RUN your program and correct its errors. After you debug your program, don't forget to SAVE it again to correct the copy on your disk.

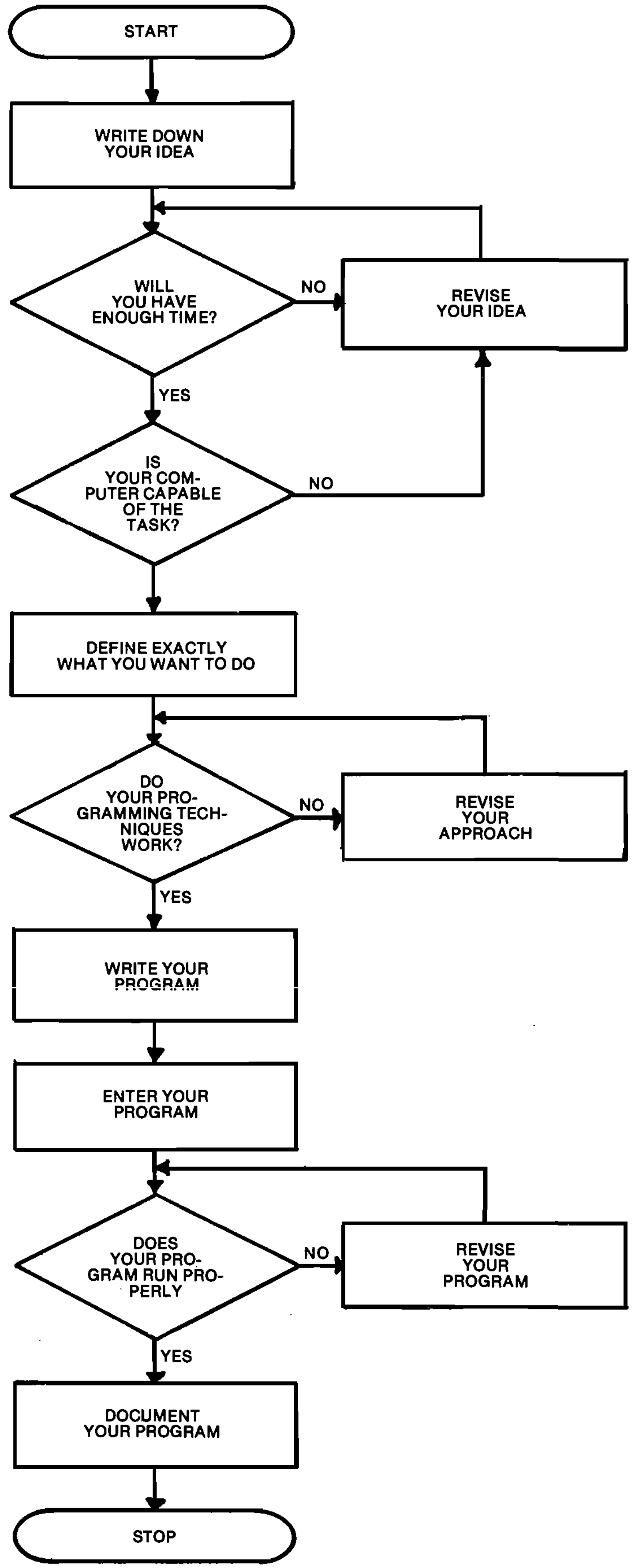


Figure 5-2
Program Planning Flowchart

Logic errors are more difficult to find and correct. These bugs can only be found by running your program and checking to make sure that it runs correctly. If it does not, you have probably made logic errors. Run your program and check its logic as follows:

(1) Input simple data that you can check by performing the computer's calculations yourself. For example you might check the logic in EASY03 (see page A-5) by entering 4 or 6 or 12. The first entry should give three factors: 1,2, and 4. The second should give 1,2,3, and 6, and the third should give 1,2,3,4,6, and 12.

(2) Input data that will test out all the logical paths in your program. That is, make sure that all your branches are executed properly. In EASY03, for example, you would want to check that line 560 is executed only when the relationship in line 520 is true. You would also want to be sure that the program terminates when the word "QUIT" is entered (lines 440 - 460).

Correct as many logic errors in your program as you can find. Once this is done, give your program to a friend to try. (Other people always seem to be very good at finding mistakes!)

Document your program. Program documentation consists of written instructions to others on how to use your program. This guide is the CLASSIC documentation, and you would have a hard time using CLASSIC without it. The same is true of the programs that you write: they will be difficult for others to use without documentation. You will use your work from this module to learn about program documentation in Module 5-C.

SELF-TEST

Discuss your program plan and final product with your instructor or system manager. He or she should agree that your program accomplishes all of the tasks outlined in your plan efficiently. If any tasks have been left out or if more efficient programming techniques are warranted, incorporate these into your program before going on to Module 5-C.

MODULE 5-C

DOCUMENTING YOUR PROGRAMS

WHAT YOU WILL DO

Document the program that you developed for Module 5-B so that it may be used by someone else without personal assistance from you.

HOW FAR YOU SHOULD GO

At least two of your friends or colleagues should be able to use your program without any assistance.

THINGS YOU WILL NEED

(1) Program developed for Module 5-B
(2) CLASSIC
(3) BASIC System disk

WHAT IT'S ALL ABOUT

Program documentation has two parts: directions for running and using the program and information to allow others to change the program. Documenting your program, though sometimes difficult, is always worthwhile, because it makes it much easier for others to use your program. This module deals with documenting your programs so that they may be used by others on the CLASSIC. (The next module examines the problem of writing your programs so that they may be used on computers other than CLASSIC.)

Directions. Whenever you begin to document a program, ask yourself this question: "If I hand a disk with my program on it to some friends who know how to operate CLASSIC, what additional information will they need to run my program?" First, others will need to know the **name of your program** so that they can call it into memory with the BASIC editor OLD command. Second, they will need a **statement of your program's purpose** so that they know what to expect the program to do. As soon as they RUN your program, other people might find that they do not understand its directions for supplying input. Therefore, the third documentation need is **a description of how the program works.** Fourth, make it a practice to **warn other users in advance** about possible bugs, limitations, or other unusual things that they might find when they use the program. Finally, give them **a complete listing** of your program so that they can make corrections if anything goes wrong.

To summarize, it is recommended that you give others the following five pieces of information to help them use a program that you have written:

(1) the name of your program,
(2) a statement of its purpose,
(3) a description of how it works,
(4) a warning about possible errors or limitations, and
(5) a complete listing.

The write-up for MORGAG on page A-10 has each of these details clearly marked.

Further information. In Module 5-B it was mentioned that you might want to modify someone else's programs rather than write a similar one from scratch. This idea can be carried one step further by saying: almost any program that you could possibly dream of writing has already been written in some form on someone else's computer! So why write new programs at all?

The answer is two-fold. First, programs written on one computer are often difficult to use on another computer for various reasons. (The ability of a program to run on more than one computer is called its **transportability,** and the considerations involved in writing transportable programs are discussed in the next module.) Second, other people's programs will seldom do exactly what you want them to do. You will therefore want to change or **modify** these programs, but you may find this much more difficult than it first appears if the author has not documented his or her program clearly. The following four guidelines are therefore suggested for documenting programs so that they may be easily modified.

(1) Use REMark statements freely.
(2) Leave large differences between the line numbers.
(3) Include a variable directory.
(4) Program in a modular fashion.

Each of these guidelines is discussed more fully below, and documentation for each program mentioned appears in Appendix A.

Use REMark statements freely. Program QUADEQ is a relatively simple program. Look at the listing for this program on page A-11. This is a good program because it performs a difficult calculation and can handle most quadratic equations of the form $Ax^2 + Bx + C = 0$.[1] But suppose you wanted to modify this program so that it would print a special message if the equation only had one root. Or perhaps you would like it to graph the equation or identify "degenerate" quadratics (where A=0). Even though QUADEQ is simple, you would have to spend some time figuring out how this program works before you could modify it. This job would be easier if the programmer had included REMark statements in his program as guides. Look at the listing of program QUAD02 on page A-12. This is exactly the same as QUADEQ, except that REMark statements have been added. Which do you think would be easier to modify?

A word of caution — realize that REMark statements take up room in your program just like other BASIC statements. That is, QUAD02 will take up more storage space on your disk than QUADEQ. Other users may wish to copy your program onto a crowded disk which does not have enough room for the program with all its REMark statements. If your program never branches to a REMark, they may delete these statements without huring your program and save space on their disk. Therefore, you should never refer to the line number of a REMark statement in a GOTO, IF-THEN, IF-GOTO, or GOSUB statement.

[1]Quadratic equations are usually studied in beginning algebra. They may always be written in the form $Ax^2 + Bx + C = 0$ and have one or two "roots", which are values that may be substituted for "x" to make the equation true.

Leave large differences between the line numbers. Look at the listing for the program HMRABI on page A-6. This program has a good number of REMark statements, but it is still quite difficult to modify because there are often very small differences between line numbers. For example, there isn't much chance of modifying the program between lines 320 and 324 because each line number is used. The solution to this problem is simple: use the RESEQ program on your CLASSIC System disk (see page 3-39) to resequence the lines and leave enough room for modification. You might try starting your programs with line 1000 and using a step of 10. This leaves plenty of room for adding BASIC statements.

Include a variable directory. Program QUADEQ was made easier to understand by adding REMark statements to make it into QUAD02. This program can be made even clearer by adding a **variable directory** as was done in QUAD03 (see page A-12). This directory lists each variable in the program and tells what each is used for. It provides a quick reference for "decoding" the BASIC language program.

Another illustration of the variable directory's value is in program HMRABI. In the write-up for this program on page A-6, you will find three hints on the interrelationships of various things in the game. These relationships were discovered only after studying the program (lines 450, 455, and 540) to find out what all the variables stand for. Perhaps the original author was trying to hide these relationships from the user to make the game more dependent upon chance. You will find that HMRABI is still quite a challenge even with this extra information.

Program in modular fashion. The last guideline is both a suggestion for clear documentation and programming in general. "Program in a modular fashion" means that you should have distinct parts in your programs and make generous use of subroutines. This technique is best illustrated by programs ACEY02 and HURK02. If you compare these two programs carefully, you will see that modular programming simplified the programming as well as the documentation.

Other techniques. After studying the programs and guidelines in this module, you may still be wondering why some things were done as they were. For example, why wasn't CLASSIC's multiple statement per line capability used in ACEY02 and HURK02? Why was the word LET used in variable assignment statements even when this word is not needed with CLASSIC? The answers to these questions involve considerations of program transportability and are discussed in the next module.

SELF-TEST

Document the program that you developed for Module 5-B according to the guidelines that have been described. Give your documented program and write-up to two of your friends and colleagues and ask them to use your program for its intended purpsoe. Let them do so on their own, and then discuss their experiences. If your program is properly documented, your friends should have been able to run it without assistance and give you valuable comments on your work.

MODULE 5-D

TRANSPORTING YOUR PROGRAMS

WHAT YOU WILL DO

Make the programs that you wrote for Module 5-B transportable to other computers.

HOW FAR YOU SHOULD GO

Submit the final version of your program and its documentation (developed in Module 5-C) to the Digital Equipment Computer Users Society (DECUS) in acceptable form.

THINGS YOU WILL NEED

(1) Program developed for Module 5-B
(2) Documentation written in Module 5-C
(3) CLASSIC
(4) CLASSIC System disk

WHAT IT'S ALL ABOUT

There are several points that you should consider in writing transportable programs[1]. These are:

(1) REMark statements,
(2) multiple statements per line,
(3) terminal display characteristics,
(4) the LET statement, and
(5) statement numbers.

REMark statements. In Module 5-C, it was recommended that you use REMark statements freely. In this module, this recommendation is reinforced to draw your attention to another use of the REMark statement: making specific comments on specific program statements. This was done in programs ACEY02 and HURK02 to make them readily adaptable to the DECsystem-10, one of DIGITAL'S larger computers. As a matter of fact, these programs were originally written on the DECsystem-10 and then transferred to CLASSIC. Thus you can see that careful planning can make your BASIC language programs relatively easy to transport from one computer to another. Remember, however, that you should never branch to REMark statements so that other users can delete them without creating other problems in your program. Look over your program and add more REMark statements if necessary.

Multiple statements per line. For each program statement that you begin with a line number, a certain amount of **overhead** is taken up in CLASSIC's memory and in the disk storage file. That is, the statement:

100 PRINT \ PRINT \ PRINT "HELLO!"

will take up less room than:

[1]For further guidelines on transportability, read the following two articles:

(a) Confer, Ronald W. "Universal BASIC: A Way to Reduce Conversion Costs". *ACM SIGCUE Bulletin* 8(2):3-9, April 1974. (The address of the ACM SIGCUE is given in Module 5-E.)

(b) Isaacs, Gerald L. "Interdialect Translatability of the BASIC Language". *ACM SIGCUE Bulletin* 8(4):11-22, October 1974. 1974.

100 PRINT
110 PRINT
120 PRINT "HELLO!"

because the first way has fewer line numbers. While it is highly desirable to write your programs so that they are as compact and efficient as possible, unfortunately many computer systems do not allow multiple statements per line. Multiple statements on one line also make programs much harder to revise. Thus programs ACEY02 and HURK02 do not contain multiple statements per line because they were written to be very transportable. Separate any multiple statements that you have in your program and use the RESEQ program to provide additional space between line numbers. Then RUN your program to make sure that it still works.

Terminal display characteristics. There are two characteristics of your terminal (keyboard/screen) that you will want to consider in writing transportable programs: line width and display speed.

The screen on your CLASSIC has a width of 80 columns. Most terminals, however, have widths of only 72 columns. For this reason, you should limit the lengths of your program and output lines to 72 characters, or others will get peculiar-looking displays when they try to RUN or LIST your program on a smaller screen. (Your statements **must** be limited to 72 characters if you plan to use the RESEQ program.)

You are probably already aware that your screen displays information very quickly, and perhaps you have already learned a few tricks for using the 12 lines on your screen without losing information at the top. Other terminals, however, often print at 1/6 the speed of your screen, a rather slow pace for the average reader. Thus, if you are developing highly interactive programs, you should keep the print-out to a minimum. If you need to give very long directions or diagrams, include them in your documentation.

Revise your program with these two considerations to improve their transportability.

The LET statement. CLASSIC does not require you to use the word LET in statements such as:

*200 D = B ∧ 2 - 4*A*C*

Most newer BASIC systems do not require the word LET either, but some of the older ones still do. The interesting point is that **all** BASIC language systems **allow** the word LET, even if it is not required. Therefore, if you use LET your statement will be understood by all BASIC language computers, while omitting LET will cause a problem with some.

Similarly, CLASSIC allows either ** or ∧ to represent exponentiation. The ∧ is more often available on other systems, so you should use this notation in your programs. Make these modifications before you go on.

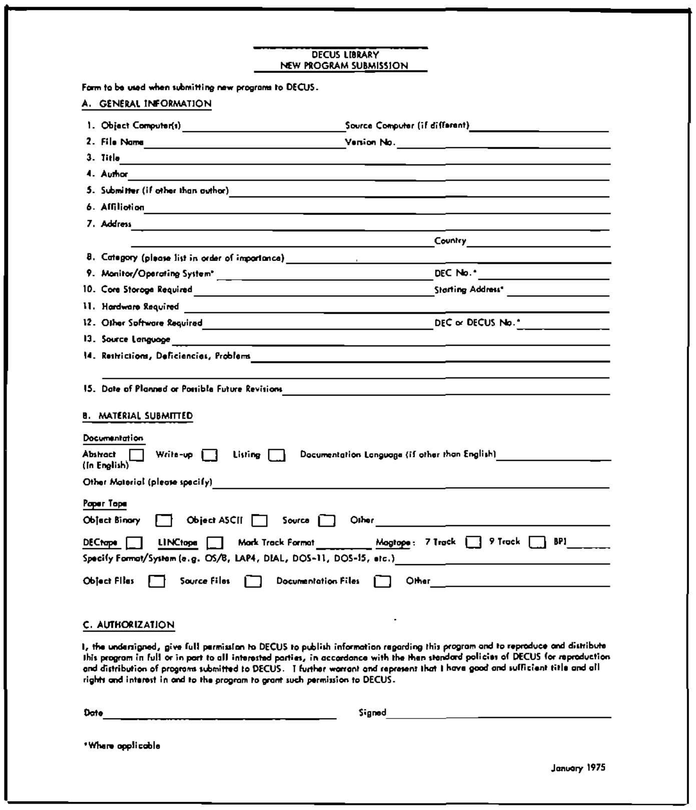
Statement numbers. Your computer system allows statements to be numbered up to 99999. One of DIGITAL's other computers (the PDP-11) only allows statements up to 32767, and some smaller machines can only accept numbers to 9999. Most BASIC language systems allow statement numbers up to 9999. Thus, try to keep your statement numbers to 4 digits so that they will run on any machine. This is easily done with the RESEQ program after your program is fully developed.

Revise and test the program that you developed for Module 5-C according to the transportability guidelines stated above.

The DIGITAL Equipment Computer Users Society (DECUS) is a company-supported organization for users of DIGITAL computers. One of the major functions of the society is the DECUS Library which distributes all types of programs. These programs are submitted by users of DIGITAL computers and are distributed to other users upon request. A minimal charge is made for handling and reproduction.

To submit the program that you wrote in Module 5-C, fill out a New Program Submission form and send it to DECUS with your program. A copy of this form appears in Figure 5-3. (Another copy is in Appendix B and additional forms may be obtained by writing to DECUS at one of the addresses listed on the form.) To fill out this form, follow these steps:

1. **Object Computer** - the computer that your program is intended to run on. In most cases, this will be CLASSIC for the program that you write. (The **source computer** is the computer that the program was written on; also CLASSIC for most of your programs.)
2. **File Name and Version No.** - the actual name that you have used to SAVE your program on a disk (e.g., WTDAVG) and the version number that you have assigned to it (if any).
3. **Title** - the full title of your program, e.g., Weighted Averaging.
4. **Author** - your name or the name of the person who wrote the program.
5. **Submitter** - your name (if you are not the author).
6. **Affiliation** - the name of your school, college, or organization.
7. **Address** - the address of your school, college, or organization.
8. **Category** - a one or two word classification of your program such as the areas of instructional computer applications discussed in Module 5-A.
9. **Monitor/Operating System** - "OS/8" for all programs written on CLASSIC.
10. **Core Storage Required** - write "less than 16K", meaning that your program will fit on a CLASSIC system.
11. **Hardware Required** - if your program refers to the disk drives, screen, or printer by name (RXA0, RXA1, TTY, or LPT), write the name of the required device here.
12. **Other Software Required** - if you are using a standard CLASSIC system, write "none". Otherwise indicate the required software here.


DECUS LIBRARY
NEW PROGRAM SUBMISSION

Form to be used when submitting new programs to DECUS.

A. GENERAL INFORMATION

1. Object Computer(s) ____ Source Computer (if different) ____
2. File Name ____ Version No. ____
3. Title ____
4. Author ____
5. Submitter (if other than author) ____
6. Affiliation ____
7. Address ____
   ____ Country ____
8. Category (please list in order of importance) ____
9. Monitor/Operating System* ____ DEC No.* ____
10. Core Storage Required ____ Starting Address* ____
11. Hardware Required ____
12. Other Software Required ____ DEC or DECUS No.* ____
13. Source Language ____
14. Restrictions, Deficiencies, Problems ____
15. Date of Planned or Possible Future Revisions ____

B. MATERIAL SUBMITTED

Documentation

Abstract (In English) ☐ Write-up ☐ Listing ☐ Documentation Language (if other than English) ____

Other Material (please specify) ____

Paper Tape

Object Binary ☐ Object ASCII ☐ Source ☐ Other ____

DECtape ☐ LINCtape ☐ Mark Track Format ____ Magtape: 7 Track ☐ 9 Track ☐ BPI ____

Specify Format/System (e.g. OS/8, LAP4, DIAL, DOS-11, DOS-15, etc.) ____

Object Files ☐ Source Files ☐ Documentation Files ☐ Other ____

C. AUTHORIZATION

I, the undersigned, give full permission to DECUS to publish information regarding this program and to reproduce and distribute this program in full or in part to all interested parties, in accordance with the then standard policies of DECUS for reproduction and distribution of programs submitted to DECUS. I further warrant and represent that I have good and sufficient title and all rights and interest in and to the program to grant such permission to DECUS.

Date ____ Signed ____

*Where applicable

January 1975

Figure 5-3
DECUS New Program Submission Form

13. **Source Language** - the language in which your program is written; BASIC for all programs written on a standard CLASSIC system.
14. **Restrictions, Deficiencies, Problems** - describe here any special characteristics that are important for people to understand when using your program, such as limits on array size.
15. **Date of Planned or Possible Future Revisions** - if you plan to modify your program, indicate when on this line.
16. **Documentation** - you must always submit listings for your programs or they will not be accepted. Short and well-documented programs may only require a short abstract to make them understandable, but longer programs should be submitted with complete write-ups such as those in Appendix A. If you send your program on a disk, write "RX01 flexible disk" on the *Other Material* line.
17. The other materials listed do not apply to CLASSIC and should be left blank.
18. Sign and date the authorization statement at the bottom of the form.

If your program is less than 100 lines long, you need to send only the DECUS form and a LISTing of it for acceptance. If possible, however, send DECUS an actual disk with your program on it. (Your disks may be mailed safely, but label them "MAGNETIC MATERIALS — DO NOT X-RAY".) DECUS will copy your program and return your disk or replace it with a new one. If your program is over 100 lines long, you **must** send it on a disk for it to be accepted. If you revise your program at a later date, you should submit a Program Revision Submission form (see Appendix B).

SELF-TEST

The self-test for this module is simply to submit your documented, transportable program to DECUS in acceptable form. You may photocopy the New Program Submission form in Appendix B or write to DECUS for more forms. Fill out the form completely, and send your program to DECUS on a disk if you can. If you are short of disks, collect programs from several of your friends onto one disk and submit them together.

MODULE 5-E

IDENTIFYING FURTHER RESOURCES

WHAT YOU WILL DO

Identify books, magazines, and organizations that can provide you with information on instructional computing beyond that given in this Guide.

HOW FAR YOU SHOULD GO

Identify at least five resources that can provide you with further information as described above.

WHAT IT'S ALL ABOUT

The following is a short list of books, magazines, and organizations that you may wish to read, subscribe to, and contact for further information on instructional computing.

Books. The number of books in print that deal specifically with instructional computer applications is relatively small. On the other hand, a large number of books have been written on the BASIC language, and many of these contain small application programs as examples. A review of 34 books on BASIC is published in a serialized article that began in the March-April issue of *Creative Computing* (see page 5-13 for the address of this magazine). The titles of these books are listed below in the order in which they were published.

Title	Author	Publisher
1. BASIC, Sixth Edition	Waite and Mather	University Press of N.E.
2. BASIC Programming	Kemeny and Kurtz	Wiley
3. Programming in BASIC	Farina	Prentice Hall
4. Introduction to an Algorithmic Language (BASIC)	(no author)	NCTM
5. Introduction to Computing Through BASIC Language	Nolan	Holt, Rinehart & Winston
6. A Guide to BASIC Programming	Spencer	Addison-Wesley
7. Problem-Solving With The Computer	Sage	Entelek
8. Introduction to Programming: A BASIC Approach	Hare	Harcourt-Brace
9. BASIC For Beginners	Gateley and Bitter	McGraw-Hill
10. Discovering BASIC	Smith	Hayden
11. Basic BASIC	Coan	Hayden
12. Computer Science: BASIC Language Programming	Forsythe, et al.	Wiley
13. Elementary BASIC With Applications	Farina	Prentice-Hall
14. Teach Yourself BASIC	Albrecht	Tecnica
15. Time Sharing's BASIC Language		General Electric
16. BASIC Programming	Murrill and Smith	Intext
17. BASIC: An Introduction to Computer Programming...	Sharpe and Jacob	Free Press
18. Computer Programming in BASIC	Pavlovich and Tahan	Holden-Day
19. An Introduction to the BASIC Language	Skelton	Holt, Rinehart & Winston
20. Basic BASIC: Self-Instructional Manual	Peluso, et al.	Addison-Wesley
21. BASIC Programming for Business	Sass	Allyn & Bacon
22. Fundamental Programming Concepts	Gross and Brainerd	Harper & Rowe
23. Programming Time-Shared Computers in BASIC	Barnett	Wiley
24. Introducing BASIC	Blakeslee	Educomp
25. Computing with the BASIC Language	Gruenberger	Canfield Press
26. Business Programming with BASIC	Diehr	Wiley
27. Entering BASIC	Sack and Meadows	SRA
28. My Computer Likes Me	Albrecht	Dymax
29. Elements of BASIC	Lewis and Blakeley	NCC
30. A Visual Approach to BASIC	Smith	CDC
31. BASIC, A Computer Programming Language...	Pegels	Holden-Day
32. BASIC	Albrecht, Finkel & Brown	Wiley
33. A Guided Tour of Computer Programming in BASIC	Dwyer and Kaufman	Houghton Mifflin
34. Principles of Data Processing	Stern and Stern	Wiley

Some of the following may also be of interest to you.

Atkinson, Richard C., and H. A. Wilson. *Computer-Assisted Instruction: A Book of Readings.* Academic Press, Inc., New York, N.Y. 1969.

Ball, Marion J. *What Is a Computer?* Houghton Mifflin Company, Boston, Mass. 1972. (Elementary level.)

Holtzman, Wayne H. (ed.) *Computer-Assisted Instruction, Testing, and Guidance.* Harper & Row, Publishers, New York, N.Y. 10016. 1970.

Lippey, Gerald (ed). *Computer-Assisted Test Construction.* Educational Technology Publications, Englewood Cliffs, N.J. 07632. 1974.

Martin, James. *Design of Man-Computer Dialogues.* Prentice-Hall, Inc., Englewood Cliffs, N.J. 1973.

Nelson, Theodor H. *Computer Lib/Dream Machines.* Hugo's Book Service, Box 2622, Chicago, Illinois 60690. 1974.

Many additional short books and pamphlets are available at low cost from DIGITAL. Write to Digital Equipment Corporation, Communications Services, Marlboro, Massachusetts 01752 to order a *Curriculum Materials Product Catalog.* Some representative materials are listed below.

Problems for Computer Mathematics
Advanced Problems for Computer Mathematics
101 BASIC Computer Games
Understanding Mathematics and Logic Using BASIC Computer Games
BASIC Matrix Operations
Computer-Augmented Calculus Topics
A Curriculum Guide for a School Computer Program in Mathematics
Huntington I Simulation Programs

Huntington II Simulation Programs
Getting Started in Classroom Computing
Population: A Self-teaching BASIC Primer
A Curriculum Guide for Teaching BASIC
Business Data Processing I
Business Data Processing II
Computer Problems for Business

Magazines. There are several magazines that contain articles on instructional computing. Some representative publications are listed by the addresses from which they can be ordered.

Creative Computing
Ideametrics
P. O. Box 789-M
Morristown, N.J. 07960

EDU
Educational Products Group
Digital Equipment Corp.
146 Main Street
Maynard, Mass. 01754

Educational Technology
Educational Technology Publications
140 Sylvan Avenue
Englewood Cliffs, N.J. 07632

People's Computer Company
P. O. Box 310
Menlo Park, CA 94205

THE Journal (Technical Horizons in Education)
Information Synergy, Inc.
P. O. Box 992
Acton, Mass. 01720

Organizations. The following organizations address themselves directly to the topic of applying the computer effectively in instructional situations. All have conventions at various locations in the United States once or twice per year. In addition, most publish a journal or newsletter and have special membership rates for students.

ACM SIGCUE (Association for Computing Machinery Special Interest Group for Computer Uses in Education)
1133 Avenue of the Americas
New York, N.Y. 10035

ADCIS (Association for the Development of Computer-based Instructional Systems)
P. O. Box 70189
Los Angeles, California 90070

AEDS NAUCAL (Association for Educational Data System, National Association for Users of Computer Applications to Learning)
1201 16th Street, N.W.
Washington, D.C. 20036

DECUS (Digital Equipment Computer Users Society)
Digital Equipment Corporation
Parker Street, Bldg. PK3-1
Maynard, Mass. 01754

This module has supplied you with a short list of resources on instructional computing. By looking at some of these resources you will find still others that you may wish to explore. Find out what books and magazines are available in your library that contain information or instructional computing and write to one or more of the organizations listed for further information. When you have identified and explored at least five additional resources, you will have completed this module.

USING THE LINE PRINTER

A. PRINTING FILES

1. From the monitor, use the TYpe Command:

   The TYpe command is normally used to display the contents of printable files on the screen. By specifying LPT: as the output device, the file is transmitted instead to the line printer.

   .TY LPT: HURKLE.BA ---Prints the BASIC program "HURKLE" on line printer.

2. From the BASIC editor, use the SAVE Command:

   The SAVE command under BASIC is normally used to store (save) BASIC language programs on floppy disks. You can, however, direct BASIC to save programs on the line printer by typing:

   .SAVE LPT:

B. PRINTING FILE DIRECTORIES

To print a directory on the line printer, specify LPT: as the output device:

.DIR LPT: ---prints the entire directory of RXA0 on the line printer

.DIR LPT: RXA1: ---lists the directory of disk #1 on the line printer

C. ACCESS TO LINE PRINTER FROM BASIC PROGRAMS

BASIC language programs can send output to the printer by using the FILEV and PRINT# statements. The PRINT statement normally writes data on the screen. However, by using the FILEV# statement to open the "LPT:" file, output is directed to the line printer.

The PRINT# statement writes data onto the line printer and is of the form:

(line number) PRINT#N: list of expressions and delimiters

where N is a numerical expression equated to the line printer. The expressions in the list can be string or numeric, and the TAB and PNT functions can both be used. The delimiters can be commas or semicolons and have the same meanings that they have in the PRINT statement for the terminal.

```
10 FILEV#1:"LPT:"
20 LET F=1
30 PRINT#F:TAB(28);DAT$(X)
40 CLOSE#F
50 END
```

This routine prints the date, starting at column 28 on the line printer.

NOTE: The user must CLOSE# all output file before ending the program in order to prevent the loss of data.

Appendix A
Write-Ups for Applications Programs

APPLICATIONS PROGRAMS

The programs that are written up in this Appendix all reside on the BASIC Program Demonstration disk. The write-ups are presented in the following order:

	Page
ACEY02	A-1
ATTEND and ATTSET	A-3
CALC	A-5
EASY02 and EASY03	A-5
GUESS	A-6
HMRABI	A-6
HURKLE	A-7
HURK02	A-8
MORGAG	A-10
QUADEQ, QUAD02, and QUAD03	A-11
SYNONY and SYNSET	A-12
WTDAVG	A-13

Sample runs for some of these programs appear at the end of Chapter 1.

ACEY02

This program allows you to play the card game "ACEY-DEUCEY" with the computer. CLASSIC will "deal" two cards and then ask you to bet on whether a third card will fall **between** the other two. For example, if your first two cards are a nine and a queen, you will win if the third card is a ten or jack. If any other card turns up (including a nine or a queen) you will lose. When you win, the amount of money that you bet is added to your pocket; when you lose it is subtracted.

The program has three options which are explained in the instructions within the program itself. If you do not put in a valid number to the YOUR BET ($)? query, the system will interpret each character that is non-numeric as a 0. When you finally get the ? back and the system is waiting for your input, retype your bet as a valid number. Some interesting things happen if you try to make negative bets.

The program might be modified by adding a "probability" option which would allow the user to request a computation of the probability of winning with any given hand. Class discussion of win/loss probabilities would increase the program's educational value.

```
LIST

ACEY02  BA    3.0     30-DEC-75

1000 REM           * * * * * * * * * * * * * * * * * * * * * *
1010 REM           *                                         *
1020 REM           *     A C E Y - D U C E Y    T W O        *
1030 REM           *                                         *
1040 REM           * * * * * * * * * * * * * * * * * * * * * *
1050 REM
1060 REM
1070 REM
1080 REM
1090 REM
1100 REM
1110 REM
1120 REM
1130 REM
1140 REM
1150 REM
1160 REM
1170 REM
1180 REM
1190 REM    * * * * * * *   V A R I A B L E   D I R E C T O R Y
1200 REM
1210 REM    VARIABLE         USAGE
1220 REM    --------         -----
1230 REM
1240 REM       A        INPUT CODE:  0='YES', 1='NO'
1250 REM       A$       GENERAL ALPHAMERIC USER INPUT
1260 REM       B        USER'S BET
1270 REM       C0$      CHR$(34)  ["]
1280 REM       C1$      C0$ & ", " & C0$  [", "]
1290 REM       D(K)     CARDS DEALT
1300 REM       K        GENERAL FOR-NEXT LOOP INDEX
1310 REM       N$(K)    CARD NAMES
1320 REM       R        ROUND COUNTER
1330 REM       R$(K)    ROUND NAMES:  R(0)="FIRST", R$(1)="NEXT"
1340 REM       X        NEGATIVE BET COUNTER
```

continued on next page

```
1350 REM
1360 REM
1370 REM    * * * * * * *    D E C L A R A T I O N S
1380 REM
1390 LET CO$=CHR$(34)
1400 LET C1$=CO$ & ", " & CO$
1410 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
1420 REM          LET C1$=CO$+", "+CO$
1430 DIM D(3),N$(14,8),R$(2,8)
1440 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
1450 REM          DIM D(3),N$(14),R$(2)
1460 DATA "   ", "DEUCE", "THREE", "FOUR", "FIVE", "SIX", "SEVEN"
1470 DATA "EIGHT", "NINE", "TEN", "JACK", "QUEEN", "KING", "ACE"
1480 FOR K=1 TO 14
1490 READ N$(K)
1500 NEXT K
1510 LET B=.01
1520 LET M=100
1530 LET R=0
1540 LET R$(0)="FIRST"
1550 LET R$(1)="NEXT"
1560 LET X=0
1570 DEF FNA(X)=2+INT(13*RND(0))
1580 REM
1590 REM                     ------------------------
1600 REM    * * * * * * *    M A I N    P R O G R A M
1610 REM                     ------------------------
1620 REM
1630 GOSUB 2620
1640 PRINT "ACEY-DUCEY TWO"
1650 PRINT "--------------"
1660 PRINT
1670 PRINT
1680 PRINT "DO YOU WISH TO SEE THE INSTRUCTIONS ("; CO$; "YES"; CO$;
1690 PRINT " OR "; CO$; "NO"; CO$; ")";
1700 GOSUB 2670
1710 IF A=0 THEN 1730
1720 GOSUB 2860
1730 GOSUB 3270
1740 GOSUB 3200
1750 REM
1760 REM    * * * * * * *    D E A L E R
1770 REM
1780 FOR K=1 TO 3
1790 LET D(K)=FNA(K)
1800 NEXT K
1810 PRINT
1820 PRINT "HERE ARE YOUR "; R$(R); " TWO CARDS..."
1830 PRINT N$(1); N$(D(1))
1840 PRINT N$(1); N$(D(2))
1850 PRINT
1860 PRINT "YOUR BET ($)";
1870 LET R=1
1880 REM
1890 REM    * * * * * * *    B E T    I N P U T
1900 REM
1910 INPUT B
1920 PRINT
1930 IF B<0 THEN 2280
1940 IF B=0 THEN 2350
1950 IF B=77777 THEN 2330
1960 IF B=88888 THEN 2410
1970 IF B=99999 THEN 3930
1980 IF B>M THEN 2500
1990 REM
2000 REM  *******  DEALING OF THIRD CARD
2010 REM
2020 PRINT "YOUR THIRD CARD IS..."
2030 PRINT N$(1); N$(D(3))
2040 PRINT
2050 IF D(3)=D(1) THEN 2140
2060 IF D(3)=D(2) THEN 2140
2070 IF D(3)>D(1) THEN 2100
2080 IF D(3)>D(2) THEN 2210
2090 GOTO 2140
2100 IF D(3)<D(2) THEN 2210
2110 REM
2120 REM  *******  PLAYER LOST
2130 REM
2140 PRINT "SORRY, YOU LOSE."
2150 LET M=M-B
2160 GOSUB 3250
2170 GOTO 1780
2180 REM
2190 REM  *******  PLAYER WON
2200 REM
2210 PRINT "YOU WIN!!"
2220 LET M=M+B
2230 GOSUB 3250
2240 GOTO 1780
2250 REM
2260 REM  *******  NEGATIVE BET
2270 REM
2280 GOSUB 3480
2290 GOTO 1780
2300 REM
2310 REM  *******  RESHUFFLE OPTION
2320 REM
2330 GOSUB 3170
2340 B=0
2350 PRINT
2360 GOSUB 3250
2370 GOTO 1780
2380 REM
2390 REM  *******  INSTRUCTIONS OPTION
2400 REM
2410 GOSUB 2860
2420 LET B=0
2430 GOSUB 3270
2440 PRINT
2450 PRINT "YOUR CARDS WERE..."
2460 GOTO 1830
2470 REM
2480 REM  *******  BET > MONEY
2490 REM
2500 PRINT "YOU ONLY HAVE  $"; M; " TO BET!!  TRY AGAIN."
2510 PRINT
2520 PRINT "YOUR CARDS ARE..."
2530 GOTO 1830
2540 REM
2550 REM                     ---------------------
2560 REM    * * * * * * *    S U B R O U T I N E S
2570 REM                     ---------------------
2580 REM
2590 REM
2600 REM  *******  SCREEN CLEARER
2610 REM
2620 PRINT PNT(27); "H"; PNT(27); "J";
2630 RETURN
2640 REM
2650 REM  *******  "YES", "NO", AND "QUIT" RESPONSE DECODER
2660 REM
2670 INPUT A$
2680 PRINT
2690 IF POS(A$,"Y",1)<>0 THEN 2790
2700 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
2710 REM          IF INSTR(1,A$,"Y")<>0 THEN 2790
2720 IF POS(A$,"N",1)<>0 THEN 2810
2730 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
2740 REM          IF INSTR(1,A$,"N")<>0 THEN 2810
2750 IF A$="QUIT" THEN 3930
2760 PRINT "     PLEASE INPUT "; CO$; "YES"; C1$; "NO"; CO$; ", OR ";
2770 PRINT CO$; "QUIT"; CO$; ".  YOUR CHOICE";
2780 GOTO 2670
2790 LET A=1
2800 RETURN
2810 LET A=0
2820 RETURN
2830 REM
2840 REM    * * * * * * *    I N S T R U C T I O N S
2850 REM
2860 GOSUB 2620
2870 PRINT "ACEY-DUCEY IS PLAYED IN THE FOLLOWING MANNER:  THE ";
2880 PRINT "COMPUTER WILL ACT"
2890 PRINT "AS DEALER AND "; CO$; "DEAL"; CO$; " TWO "; CO$; "CARDS";
2900 PRINT CO$; " TO YOU "; CO$; "FACE UP"; CO$; ".  YOU WILL ";
2910 PRINT "THEN HAVE"
2920 PRINT "THE OPTION TO BET OR NOT TO BET, DEPENDING UPON WHETHER ";
2930 PRINT "OR NOT YOU THINK"
2940 PRINT "THAT THE NEXT "; CO$; "CARD"; CO$; " WILL HAVE A VALUE ";
2950 PRINT "BETWEEN THE FIRST TWO (ACES ARE"
2960 PRINT "HIGH).  IF YOU DO NOT WISH TO BET, ENTER "; CO$; "0";
2970 PRINT CO$; ".  THE FOLLOWING THREE CODES"
2980 PRINT "MAY BE SUBSTITUTED FOR BETS TO PERFORM THE FUNCTIONS ";
2990 PRINT "INDICATED:"
3000 PRINT "                 77777           RESHUFFLE THE CARDS"
3010 PRINT "                 88888           REDISPLAY THESE ";
3020 PRINT "INSTRUCTIONS"
3030 PRINT "                 99999           END THE GAME"
3040 IF R=0 THEN 3070
3050 PRINT
3060 GOTO 3090
3070 PRINT "GOOD LUCK!  THE HOUSE WILL SPOT YOU  $ 100  TO BEGIN.  ";
3080 PRINT "I'M SHUFFLIN'..."
3090 PRINT
3100 PRINT "(TYPE "; CO$; "Y"; CO$; " AND PRESS "; CO$; "RETURN"; CO$;
3110 PRINT " WHEN YOU HAVE FINISHED READING.)  READY";
3120 INPUT A$
3130 RETURN
3140 REM
3150 REM    * * * * * * *    C A R D    R E S H U F F L E R
3160 REM
3170 PRINT
3180 PRINT "ZOOP!  THE CARDS ARE RESHUFFLED."
3190 PRINT
3200 RANDOMIZE
3210 RETURN
3220 REM
3230 REM    * * * * * * *    M O N E Y    P R I N T E R
3240 REM
3250 PRINT
3260 GOTO 3280
3270 GOSUB 2620
3280 PRINT "YOU ";
3290 IF B=0 THEN 3320
3300 PRINT "NOW";
3310 GOTO 3330
3320 PRINT "STILL";
3330 PRINT " HAVE  $"; M; "."
3340 IF M<=0 THEN 3390
3350 RETURN
3360 REM
3370 REM  *******  OUT OF MONEY
3380 REM
3390 PRINT
3400 PRINT
3410 PRINT "YOU BLEW YOUR WAD!  SINCE THE HOUSE SPOTTED YOU  $ 100  ";
3420 PRINT "TO BEGIN THE"
3430 PRINT "GAME, ";
3440 GOTO 4020
3450 REM
3460 REM    * * * * * * *    N E G A T I V E    B E T    T H R E A T
3470 REM
3480 LET X=X+1
3490 IF X=2 THEN 3660
3500 IF X=3 THEN 3780
3510 REM
3520 REM  *******  FIRST TIME
3530 REM
3540 PRINT "NO NEGATIVE BETS ALLOWED!  THIS TIME WE'LL LET YOU OFF ";
3550 PRINT "WITH A FINE"
3560 PRINT "OF  $"; ABS(B); ".  TRY THAT AGAIN AND WE'RE GONNA MAKE ";
3570 PRINT "YOU AN OFFER"
3580 PRINT "YOU CAN'T REFUSE!"
3590 LET M=M-ABS(B)
3600 LET B=.01
3610 GOSUB 3250
3620 RETURN
```

```
3630 REM
3640 REM  *******  SECOND TIME
3650 REM
3660 PRINT 'WADID AH TELL YA, KID?  NOW LOOKIE HERE: WE AIN'T GOT NO ';
3670 PRINT 'ROOM FA NO'
3680 PRINT 'PUNKS ON DIS KUMPOODA.  EIDER YOUSE WISES UP OR YOUSA ';
3690 PRINT 'GETSA OFF!'
3700 PRINT
3710 PRINT
3720 PRINT 'YOU NOW GOTTA ONLY ONE DOLLAH.  SPEND IT WISELY...'
3730 LET M=1
3740 RETURN
3750 REM
3760 REM  *******  THIRD TIME
3770 REM
3780 GOSUB 2620
3790 FOR K=1 TO 5
3900 PRINT 'BANG!!  '; PNT(7);
3810 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
3820 REM        PRINT 'BANG!!  '; CHR$(7);
3830 NEXT K
3840 PRINT
3850 PRINT
3860 PRINT
3870 PRINT 'SOME PEOPLE JUST NEVER LEARN...  REST IN PEACE!'
3880 PRINT
3890 GOTO 4160
3900 REM
3910 REM    * * * * * * *   Q U I T T I N G   R O U T I N E
3920 REM
3930 PRINT
3940 PRINT 'YOU ENDED UP WITH  $'; M; '.  SINCE THE HOUSE SPOTTED ';
3950 PRINT 'YOU  $ 100'
3960 PRINT 'TO BEGIN, ';
3970 IF M=100 THEN 4090
3980 IF M>100 THEN 4140
3990 REM
4000 REM  *******  THE PLAYER OWES US
4010 REM
4020 PRINT 'YOU OWE US  $'; 100-M; '.  JUST SEND YOUR CHECK TO US IN ';
4030 PRINT 'MAYNARD,'
4040 PRINT 'MASSACHUSETTS AND HAVE A NICE DAY!'
4050 GOTO 4160
4060 REM
4070 REM  *******  THE PLAYER IS EVEN
4080 REM
4090 PRINT 'YOU'RE EVEN!  GO HAPPILY, MY FRIEND.'
4100 GOTO 4160
4110 REM
4120 REM  *******  WE OWE THE PLAYER
4130 REM
4140 PRINT 'THE HOUSE OWES YOU  $'; M-100; '.  PLEASE CONTACT OUR'
4150 PRINT 'RUBBER DIVISION FOR PAYMENT.'
4160 PRINT
4170 PRINT 'BYE!'
4180 PRINT
4190 END
```

ATTEND and ATTSET

ATTEND demonstrates CLASSIC file usage by providing teachers with a method for entering, updating, and printing their class attendance records on the computer. The program computes simple statistics on attendance and might be modified to calculate other values needed by a school system.

ATTEND stores information on each student in a file called "ROSTER.AT" on RXA1. Depending upon which option you enter, the program will perform the following operations:

(1) Remove a student from the file ("DELETE" option).
(2) Add a student to the file ("ENTER" option).
(3) List the attendance data on each student in the file ("LIST" option).
(4) List the names of all students in the file ("ROSTER" option).
(5) Display the total attendance figures for the whole file ("SUMMARIZE" option).
(6) Allow you to supply attendance data for a school day on each student in the file. That is, you can inform the program whether the student was present, absent, or present but tardy today ("SUPPLY" option).
(7) Display all available options ("HELP" option).
(8) Terminate the program ("QUIT" option).

Each time you enter an option the program will perform the related operation. After each operation is completed (except "QUIT"), the program will ask for another option.

In order to use this program you must first:

(1) Enter a date into the system (with the monitor "DATE" command).
(2) Set up the file "ROSTER.AT" by running ATTSET.BA. You should only have to do this once, unless ROSTER.AT becomes unusable for some reason. The file created by ATTSET.BA will contain only the current date. If you try to run ATTEND without having set up ROSTER.AT, a message of the form "EN AT LINE nnnnn" will be displayed.

You can then use the "ENTER" option to put the names of all your students into the file ROSTER.AT. If new students join the class later, they can also be added to the file with the "ENTER" option. If students leave, they can be removed from the file with the "DELETE" option.

At this point, the file contains the name of each of your students in alphabetical order. To supply each day's attendance data for the students, use the "SUPPLY" option. The file will then contain the following data for each student:

(1) name,
(2) number of days present,
(3) number of days absent, and
(4) number of days tardy.

This is the student's **record.** A record is part of a file and is a collection of related data items treated as a unit. The data in file ROSTER.AT is arranged like this:

DATE	LAST NAME	FIRST NAME	DAYS PRESENT	DAYS ABSENT	DAYS TARDY	LAST NAME	FIRST NAME	DAYS PRESENT	DAYS ABSENT	DAYS TARDY	

First Student's Record   Second Student's Record

The "LIST" option will display all the data in the file, along with the following figures:

(1) Days registered (days present plus days absent)
(2) Percent present (days present divided by days registered)
(3) Percent absent (days absent divided by days registered)
(4) Percent tardy (days tardy divided by days present)

If the data in ROSTER.AT are not correct, you can re-create the file with ATTSET. However, since ATTSET erases all the data in the file, you will have to supply every student's attendance for each day registered to bring the file up to date.

```
LIST

ATTEND  BA      3.0      30-DEC-75

1000 REM                 -----------
1020 REM       ****      A T T E N D
1040 REM                 -----------
1060 REM
1080 REM
1100 REM
1120 REM
```

continued on next page

```
1140 REM    ****    VARIABLE DIRECTORY
1160 REM
1180 REM    VARIABLE        USAGE
1200 REM    --------        -----
1220 REM       A$        DATE READ FROM FILE
1240 REM       B$        ANSWER TO INSTRUCTIONS QUERY
1260 REM       C$        ANSWER TO OPTIONS QUERY
1280 REM       D$        ANSWER TO READY QUERY
1300 REM       E$        'RXA1:ROSTER.AT'
1320 REM       F$        LAST NAME-READ FROM FILE
1340 REM       G$        FIRST NAME-READ FROM FILE
1360 REM       H         DAYS PRESENT
1380 REM       H2        TOTAL DAYS PRESENT
1400 REM       I         DAYS ABSENT
1420 REM       I2        TOTAL DAYS ABSENT
1440 REM       J$        LAST NAME-ENTERED FROM TERMINAL
1460 REM       K$        FIRST NAME-ENTERED FROM TERMINAL
1480 REM       L         DAYS REGISTERED
1500 REM       M         PER CENT PRESENT
1520 REM       N         PER CENT ABSENT
1540 REM       O         PER CENT TARDY
1560 REM       P         DAYS TARDY
1580 REM       P2        TOTAL DAYS TARDY
1600 REM       Q$        ANSWERS TO PRESENT & TARDY QUERIES
1620 REM       R         RETURN CHARACTER-READ FROM FILE
1640 REM       S         NUMBER OF STUDENTS
1660 REM
1680 REM    ****    DECLARATIONS
1700 REM
1720 DEF FNR(X)=INT((100*X)+.5)
1740 DEF FNS(X)=LEN(STR$(INT(X)))
1760 DIM C$(72),D$(72),E$(72),F$(72),G$(72),J$(72),K$(72)
1780 DIM A$(72),B$(72)
1800 REM
1820 REM            ------------------------
1840 REM    ****    M A I N    P R O G R A M
1860 REM            ------------------------
1880 REM
1900 REM
1920 REM    ****    READ DATE RECORD FROM FILE
1940 REM
1960 LET E$='RXA1:ROSTER.AT'
1980 PRINT 'IF YOU SEE THE MESSAGE:  EN AT LINE 02120'
2000 PRINT 'BELOW, RUN THE PROGRAM ''ATTSET'' BY TYPING:'
2020 PRINT '         OLD RXA1:ATTSET'
2040 PRINT 'AND THEN:'
2060 PRINT '         RUN'
2080 PRINT
2100 PRINT 'MESSAGE:'
2120 FILE#1:E$
2140 PRINT 'NO MESSAGE'
2160 INPUT #1: A$
2180 IF POS(A$,'/',1)<=1 THEN 2320
2200 PRINT
2220 PRINT 'THE LAST TIME THAT THE ROSTER WAS UP-DATED WAS ON  '; A$
2240 GOTO 2420
2260 REM
2280 REM    ****    BAD FILE
2300 REM
2320 PRINT
2340 PRINT 'FILE ''RXA1:ROSTER.AT'' HAS BEEN CORRUPTED.  RUN PROGRAM ';
2360 PRINT'''RXA1:ATTSET.BA'''
2380 PRINT 'TO CORRECT THIS PROBLEM BEFORE USING THIS PROGRAM.'
2400 STOP
2420 PRINT
2440 REM
2460 REM    ****    CHECK SYSTEM DATE
2480 REM
2500 IF POS(DAT$(X),'/',1)>0 THEN 2580
2510 PRINT 'PLEASE RETURN TO THE MONITOR AND ENTER THE CURRENT DATE ';
2520 PRINT 'WITH THE'
2540 PRINT 'MONITOR ''DATE'' COMMAND BEFORE RUNNING THIS PROGRAM.'
2560 STOP
2580 CLOSE#1
2600 REM
2620 REM    ****    INSTRUCTIONS QUERY
2640 REM
2660 PRINT 'DO YOU WISH TO SEE THE INSTRUCTIONS (ANSWER YES OR NO)';
2680 INPUT B$
2700 IF B$='YES' GOTO 6800
2720  IF B$='NO' THEN 2820
2740 GOTO 2660
2760 REM
2780 REM    ****    OPTIONS QUERY
2800 REM
2820 PRINT 'ENTER THE OPTION YOU WANT';
2840 INPUT C$
2860 PRINT
2880 PRINT
2900 IF C$='HELP' GOTO 7220
2920 IF C$='DELETE' GOTO 3300
2940 IF C$='ENTER' GOTO 3700
2960 IF C$='LIST' GOTO 4280
2980 IF C$='ROSTER' GOTO 4960
3000 IF C$='QUIT' GOTO 3220
3020 IF C$='SUMMARIZE' GOTO 5180
3040 IF C$='SUPPLY' GOTO 5920
3060 PRINT C$;' IS NOT A VALID OPTION. THE ''HELP'' OPTION DISPLAYS A'
3080 PRINT 'LIST OF VALID OPTIONS.'
3100 GOTO 2820
3120 PRINT\PRINT
3140 PRINT C$;' PROCESSING HAS BEEN COMPLETED.  YOU MAY NOW ENTER';
3160 PRINT ' ANOTHER OPTION.'
3180 CLOSE#1
3200 GOTO 2820
3220 STOP
3240 REM
3260 REM    ****    DELETE OPTION
3280 REM
3300 GOSUB 6360
3320 PRINT 'ENTER LAST NAME OF STUDENT TO BE DELETED';
3340 INPUT J$
3360 PRINT 'ENTER FIRST NAME';
3380 INPUT K$
3400 GOSUB 6520
3420 IF END#1 THEN 3540
3440 IF J$&K$=F$&G$ THEN 3500
3460 GOSUB 6660
3480 GOTO 3400
3500 LET J$='?'
3520 GOTO 3400
3540 IF J$='?' GOTO 3600
3560 PRINT K$;' ';J$;' COULD NOT BE DELETED ';
3580 PRINT 'BECAUSE THE NAME WAS NOT IN THE FILE.'
3600 CLOSE#2
3620 GOTO 3120
3640 REM
3660 REM    ****    ENTER OPTION
3680 REM
3700 GOSUB 6360
3720 PRINT 'ENTER LAST NAME OF STUDENT TO BE ADDED';
3740 INPUT J$
3760 PRINT 'ENTER FIRST NAME';
3780 INPUT K$
3800 GOSUB 6520
3820 IF END#1 THEN 4020
3840 IF J$&K$<F$&G$ THEN 3920
3860 IF J$&K$=F$&G$ THEN 4140
3880 GOSUB 6660
3900 GOTO 3800
3920 PRINT#2:J$
3940 PRINT#2:K$
3960 PRINT#2:0,0,0
3980 LET J$='?'
4000 GOTO 3880
4020 IF J$='?' THEN 4100
4040 PRINT#2:J$
4060 PRINT#2:K$
4080 PRINT#2:0,0,0
4100 CLOSE#2
4120 GOTO 3120
4140 PRINT K$;' ';J$;' COULD NOT BE ADDED ';
4160 LET J$='?'
4180 PRINT 'BECAUSE THE NAME WAS ALREADY ON THE FILE.'
4200 GOTO 3880
4220 REM
4240 REM    ****    LIST OPTION
4260 REM
4280 FILE#1:E$
4300 INPUT#1:A$
4320 PRINT TAB(24);'ATTENDANCE DATA ';A$
4340 PRINT
4360 PRINT TAB(6);'STUDENT'S';TAB(25);'DAYS';TAB(35);'DAYS';
4380 PRINT TAB(50);'DAYS';TAB(65);'DAYS'
4400 PRINT TAB(8);'NAME';TAB(24);'REGIS.';TAB(34);'PRESENT';
4420 PRINT TAB(49);'ABSENT';TAB(65);'TARDY'
4440 GOSUB 6520
4460 IF END #1 THEN 3120
4480 LET L=H+I
4500 IF L=>1 GOTO 4560
4520 M=0\N=0\O=0
4540 GOTO 4680
4560 LET M=FNR(H/L)
4580 LET N=FNR(I/L)
4600 IF H=>1 GOTO 4660
4620 LET O=0
4640 GOTO 4680
4660 LET O=FNR(P/H)
4680 PRINT F$;' ';G$;
4700 GOSUB 4740
4720 GOTO 4440
4740 PRINT TAB(27-FNS(L));L;
4760 PRINT TAB(34-FNS(H));H;
4780 PRINT TAB(39-FNS(M));'(';STR$(M);'%)';
4800 PRINT TAB(49-FNS(I));I;
4820 PRINT TAB(54-FNS(N));'(';STR$(N);'%)';
4840 PRINT TAB(64-FNS(P));P;
4860 PRINT TAB(69-FNS(O));'(';STR$(O);'%)'
4880 RETURN
4900 REM
4920 REM    ****    ROSTER OPTION
4940 REM
4960 FILE#1:E$
4980 INPUT#1:A$
5000 PRINT TAB(4);'ROSTER    ';A$
5020 PRINT
5040 GOSUB 6520
5060 IF END#1 THEN 3120
5080 PRINT F$;' ';G$
5100 GOTO 5040
5120 REM
5140 REM    ****    SUMMARIZE OPTION
5160 REM
5180 FILE#1:E$
5200 INPUT#1:A$
5220 PRINT TAB(25);'ATTENDANCE SUMMARY ';A$
5240 PRINT
5260 LET S=0
5280 H2=0\I2=0\P2=0
5300 GOSUB 6520
5320 IF END#1 THEN 5440
5340 LET S=S+1
5360 LET H2=H2+H
5380 LET I2=I2+I
5400 LET P2=P2+P
5420 GOTO 5300
5440 LET L=H2+I2
5460 IF L=>1 GOTO 5520
5480 M=0\N=0\O=0
5500 GOTO 5640
5520 LET M=FNR(H2/L)
5540 LET N=FNR(I2/L)
5560 IF H2=>1 GOTO 5620
5580 LET O=0
5600 GOTO 5640
5620 LET O=FNR(P2/H2)
5640 PRINT TAB(6);'NUMBER OF';
5660 PRINT TAB(25);'DAYS';TAB(35);'DAYS';TAB(50);'DAYS';TAB(65);'DAYS'
5680 PRINT TAB(6);'STUDENTS';
5700 PRINT TAB(24);'REGIS.';TAB(34);'PRESENT';TAB(49);'ABSENT';
```

```
5720 PRINT TAB(65);"TARDY"
5740 LET H=H2
5760 LET I=I2
5780 LET P=P2
5800 PRINT TAB(9);S;
5820 GOSUB 4740
5840 GOTO 3120
5860 REM
5880 REM    ****    SUPPLY OPTION
5900 REM
5920 GOSUB 6360
5940 GOSUB6520
5960 IF END#1 GOTO 4100
5980 PRINT "WAS ";G$;" ";F$;" PRESENT TODAY (YES OR NO)";
6000 INPUT Q$
6020 IF Q$="YES" GOTO 6140
6040 IF Q$="NO" GOTO 6080
6060 GOTO 5980
6080 LET I=I+1
6100 GOSUB 6660
6120 GOTO 5940
6140 LET H=H+1
6160 PRINT "WAS ";G$;" ";F$;" TARDY TODAY (YES OR NO)";
6180 INPUT Q$
6200 IF Q$="YES" THEN 6260
6220 IF Q$="NO" THEN 6100
6240 GOTO 6160
6260 LET P=P+1
6280 GOTO 6100
6300 REM
6320 REM    ****    OPEN FILES
6340 REM
6360 FILE#1:E$
6380 INPUT#1:A$
6400 FILEV#2:E$
6420 PRINT#2:DAT$(X)
6440 RETURN
6460 REM
6480 REM    ****    READ A STUDENT
6500 REM
6520 INPUT#1:F$
6540 IF END#1 THEN 6580
6560 INPUT#1:G$,R,H,I,P,R
6580 RETURN
6600 REM
6620 REM    ****    WRITE A STUDENT
6640 REM
6660 PRINT#2:F$
6680 PRINT#2:G$
6700 PRINT#2:H,I,P
6720 RETURN
6740 REM
6760 REM    ****    PRINT INSTRUCTIONS
6780 REM
6800 PRINT
6820 PRINT "THIS PROGRAM MAINTAINS AN ATTENDANCE FILE."
6840 PRINT "IT ALLOWS YOU TO ENTER ATTENDANCE DATA AND PRINT ";
6860 PRINT "OUT THIS DATA"
6880 PRINT "IN A STANDARD FORM.  THE FILE USED BY THIS PROGRAM IS  ";
6900 PRINT "ROSTER.AT  AND"
6920 PRINT "IT MUST BE PRESENT ON RXA1 IN CORRECT FORM FOR THIS ";
6940 PRINT "PROGRAM TO RUN."
6960 PRINT "THIS FILE MAY BE CREATED (OR CLEARED) BY RUNNING ";
6980 PRINT "PROGRAM ATTSET.BA ON"
7000 PRINT "RXA1.  IF YOU WANT TO SEE THE LIST OF OPTIONS AVAILABLE ";
7020 PRINT "IN THIS PROGRAM,"
7040 PRINT "ENTER ""HELP"" AFTER THE OPTION QUERY."
7060 PRINT
7080 PRINT "(TYPE ""Y"" AND PRESS ""RETURN"" WHEN YOU";
7100 PRINT " HAVE FINISHED READING.)  READY";
7120 INPUT D$
7140 GOTO 2820
7160 REM
7180 REM    ****    HELP OPTION
7200 REM
7220 PRINT "THE AVAILABLE OPTIONS ARE:"
7240 PRINT "      DELETE           REMOVE A STUDENT FROM THE ROSTER"
7260 PRINT "      ENTER            ADD A STUDENT TO THE ROSTER"
7280PRINT "     HELP             DISPLAY THIS MESSAGE"
7300 PRINT "      LIST             LIST ATTENDANCE DATA ON ALL STUDENTS"
7320PRINT  "      ROSTER           LIST THE ROSTER"
7340 PRINT "      QUIT             TERMINATE THIS PROGRAM"
7360 PRINT "      SUMMARIZE        SUMMARIZE THE ATTENDANCE DATA"
7380 PRINT "      SUPPLY           INPUT ATTENDANCE DATA"
7400 GOTO 2820
7420 END
```

```
LIST

ATTSET  BA    3.0     30-DEC-75

1000 REM             -----------
1020 REM    ****     A T T S E T
1040 REM             -----------
1060 REM
1080 IF POS(DAT$(X),"/",1)>0 THEN 1200
1100 PRINT
1120 PRINT "NO DATE HAS BEEN ENTERED TO THE SYSTEM.  PLEASE DO SO ";
1140 PRINT "WITH THE"
1160 PRINT "MONITOR ""DATE"" COMMAND BEFORE YOU RUN THIS PROGRAM."
1180 STOP
1200 FILEV#1:"RXA1:ROSTER.AT"
1220 PRINT #1: DAT$(X)
1240 CLOSE #1
1260 PRINT
1280 PRINT "YOU CAN NOW RUN THE PROGRAM ""ATTEND"" BY TYPING:"
1300 PRINT "         OLD RXA1:ATTEND"
1320 PRINT "AND THEN:"
1340 PRINT "         RUN"
1360 END
```

CALC

The use of the computer as a powerful calculator has always been one of the most common instructional applications. CALC allows you to input any valid CLASSIC *numerical expression and prints out the* value of that expression.

This program uses one BASIC language program to write another, CHAINs to the newly written program, and then CHAINs back to the original one. This process is necessary because different numbers can be specified in a single expression while running a program, but you must recompile the program if you want to change the expression itself. CALC takes care of this problem automatically and allows you to enter new expressions continuously. If you make an error in entering your expression and receive an error message, simply type RUN to begin the program again.

```
LIST

CALC    BA    3.0     30-DEC-75

100 REM    ****    C A L C
110 REM
120 REM
130 REM
140 REM
150 REM
160 REM    ****    VARIABLE DIRECTORY
170 REM
180 REM  VARIABLE        USAGE
190 REM  --------        -----
200 REM     E$     USER'S INPUT
210 REM     M$     MEDIUM ON WHICH THIS PROGRAM RESIDES
220 REM
230 REM
240 REM    ****    NOTES
250 REM
260 REM  BEFORE THIS PROGRAM CAN BE USED, "M$" MUST BE SET TO THE
270 REM    DEVICE SPECIFICATION FOR THE MEDIUM ON WHICH THIS PROGRAM
280 REM    RESIDES.  (SEE LINE 370.)
290 REM
300 REM  IT IS RECOMMENDED THAT ALL REMARKS BE DELETED FROM THIS
310 REM    PROGRAM TO MINIMIZE CHAINING TIME.
320 REM
330 REM
340 REM    ****    DECLARATIONS
350 REM
360 DIM E$(72)
370 LET M$="RXA1"
380 REM
390 REM    ****    USER INPUT
400 PRINT
410 PRINT "YOUR EXPRESSION";
420 INPUT E$
430 PRINT
440 IF E$="QUIT" THEN 590
450 REM
460 REM    ****    WRITING OF FILE "COMP.BA"
470 REM
480 FILEV #1: M$ & ":COMP.BA"
490 PRINT #1: "10 DIM E$(72)"
500 PRINT #1: "20 E$="""; E$; """"
510 PRINT #1: "30 PRINT E$; "" ="";"; E$
520 PRINT #1: "40 CHAIN ""RXA1:CALC.BA"""
530 PRINT #1: "99 END"
540 CLOSE #1
550 REM
560 REM    ****    CHAINING TO FILE "COMP.BA"
570 REM
580 CHAIN M$ & ":COMP.BA"
590 END
```

EASY02 and EASY03

This program finds the factors of a given number and is on your demonstration disk in exactly the same form in which it was submitted to DECUS (the Digital Equipment Corporation User Society; see Module 5-D).

The program was modified by using some of the CLASSIC string functions and adding REMarks to create EASY03.

```
LIST

EASY02  BA    3.0      30-DEC-75

10 PRINT "NUMBER OF PROBLEMS IS";
20 INPUT N
30 FOR K=1 TO N
40 PRINT
50 PRINT "NUMBER IS";
60 INPUT X
70 PRINT "FACTORS ARE:"
80 FOR F=1 TO INT(X/2+.5)
90 IF X/F<>INT(X/F) THEN 110
100 PRINT F
110 NEXT F
120 PRINT
130 NEXT K
140 END
```

```
LIST

EASY03  BA    3.0      30-DEC-75

100 REM    ****     E A S Y 0 3
110 REM
120 REM
130 REM
140 REM
150 REM
160 REM
170 REM
180 REM
190 REM    ****     VARIABLE DIRECTORY
200 REM
210 REM    VARIABLE         USAGE
220 REM    --------         -----
230 REM       F        FACTOR INDEX
240 REM       N        NUMERICAL FORM OF USER INPUT
250 REM       N$       ACTUAL USER INPUT
260 REM
270 REM    ****     DIRECTIONS
280 PRINT
290 PRINT "EASY03"
300 PRINT "------"
310 PRINT
320 PRINT
330 PRINT "THIS PROGRAM WILL FIND THE POSITIVE FACTORS OF ANY ";
340 PRINT "NUMBER THAT"
350 PRINT "YOU ENTER.  AFTER YOU HAVE ENTERED ALL THE NUMBERS ";
360 PRINT "THAT YOU"
370 PRINT "ARE INTERESTED IN, ENTER "; CHR$(34); "QUIT";
380 PRINT CHR$(34); " TO STOP THE PROGRAM."
390 REM
400 REM    ****     NUMBER INPUT
410 PRINT
420 PRINT
430 PRINT "YOUR NUMBER";
440 INPUT N$
450 PRINT
460 IF N$="QUIT" THEN 610
470 N=VAL(N$)
480 REM    ****     FACTOR CALCULATION
490 REM
500 PRINT "THE FACTORS OF"; N; "ARE:"
510 FOR F=1 TO INT(N/2+.5)
520 IF N/F<>INT(N/F) THEN 570
530 REM
540 REM    ****     FACTOR PRINT OUT
550 REM
560 PRINT TAB(1+LEN(STR$(N))-LEN(STR$(F))); F
570 NEXT F
580 PRINT N
590 REM    ****     RECYCLE FOR NEXT NUMBER
600 GOTO 410
610 END

READY
```

GUESS

This is a simple game that challenges the user to guess a number that the computer has chosen at random between 1 and 100. After each guess, the computer gives the user a hint by telling whether the guess was too high or too low.

```
LIST

GUESS   BA    3.0     30-DEC-75

100 REM            ---------
110 REM    ****    G U E S S
120 REM            ---------
130 REM
140 REM
150 REM
160 REM
170 REM
180 REM
190 REM
200 REM
210 REM    ****    VARIABLE DIRECTORY
220 REM
230 REM  VARIABLE        USAGE
240 REM  --------        -----
250 REM      G       USER'S GUESS
260 REM      K       GUESS COUNTER
270 REM      N       THE SECRET NUMBER
280 REM      N$      USER'S FIRST NAME
290 REM
300 REM
310 REM    ****    DECLARATION
320 REM
330 DIM N$(72)
340 REM
350 REM    ****    HEADER
360 REM
370 PRINT
380 PRINT "GUESS:  THE NUMBER GUESSING GAME"
390 PRINT "--------------------------------"
400 PRINT
410 PRINT
420 PRINT "PLEASE TYPE YOUR FIRST NAME AND THEN PRESS THE RETURN KEY."
430 PRINT
440 PRINT "     WHAT IS YOUR FIRST NAME";
450 INPUT N$
460 RANDOMIZE
470 PRINT
480 PRINT "HELLO, "; N$; "!"
490 PRINT
500 PRINT
510 REM
520 REM    ****    DIRECTIONS
530 REM
540 PRINT "I AM THINKING OF A NUMBER BETWEEN  1  AND  100 ."
550 GOTO 570
560 PRINT "I AM THINKING OF ANOTHER NUMBER BETWEEN  1  AND  100 ."
570 PRINT "TRY TO GUESS WHAT IT IS.  (PRESS RETURN AFTER EACH GUESS.)"
580 PRINT
590 LET N=INT(100*RND(0))+1
600 LET K=1
610 REM    ****    GUESS INPUT
620 REM
630 PRINT "YOUR GUESS";
640 INPUT G
650 REM    ****    RIGHT ON
660 REM
670 IF G<>N THEN 820
680 PRINT
690 PRINT "THAT'S IT, "; N$; "!  YOU GUESSED THE NUMBER IN"; K;
700 PRINT "GUESSES.  ";
710 IF K>7 THEN 760
720 IF K=7 THEN 740
730 PRINT "VERY ";
740 PRINT "GOOD!";
750 PRINT
760 PRINT
770 PRINT
780 GOTO 560
790 REM
800 REM    ****    TOO LOW OR TOO HIGH
810 REM
820 PRINT "  TOO ";
830 IF G>N THEN 860
840 PRINT "LOW";
850 GOTO 870
860 PRINT "HIGH";
870 PRINT ".  GUESS AGAIN."
880 PRINT
890 LET K=K+1
900 GOTO 630
910 END

READY
```

HMRABI

Hamurabi was the name of the king in the ancient city of Sumeria. HMRABI allows you to try to fill the king's shoes in managing the economy of this ancient city by buying and selling land, feeding the people from your storehouses, and planting crops over a ten year period. You will soon learn that without studying the program, this is a difficult task. Here are three hints to get you started.

(1) It takes 20 bushels of grain to feed each person in the city each year.

(2) It takes 1 person to tend every 10 acres that you wish to plant with seed.

(3) It takes 2 bushels of grain to seed each acre that you wish to plant.

The program is included on your demonstration disk in essentially the same form that it is printed in *101 BASIC Games.* Only slight modifications have been made to make it run on CLASSIC. Suggestions for improving this program are in Module 5-D.

The uses of the variables in this program are as follows:

Variable	**Usage**
A	Current acreage
C	Number of people not starved
D	Number of acres to plant Number of people starved
D1	Total number of people starved
E	Number of bushels eaten by rats
H	Number of bushels harvested
L	Final number of acres per person
P	Current population
P1	Percent of population starved per year
Q	Number of bushels to feed people Number of bushels to buy Number of bushels to sell
S	Number of bushels of grain in store
Y	Trading rate of land in bushels per acre
Z	Year

Note that one variable stands for different things at different points in the program.

```
LIST

HMRABI  BA    3.0     30-DEC-75

10 REM    ****    HMRABI
20 REM
30 REM
80 PRINT "TRY YOUR HAND AT GOVERNING ANCIENT SUMERIA"
85 PRINT "SUCCESSFULLY FOR A 10-YR TERM OF OFFICE."\PRINT
90 RANDOMIZE\LET D1=0\LET P1=0
100 LET Z=0\LET P=95\LET S=2800\LET H=3000\LET E=H-S
110 LET Y=3\LET A=H/Y\LET I=5\LET Q=1
210 LET D=0
215 PRINT\PRINT\PRINT "HAMURABI:  I BEG TO REPORT TO YOU,"\LET Z=Z+1
217 PRINT "IN YEAR";Z;",";D;"PEOPLE STARVED,";I;"CAME TO THE CITY."
218 LET P=P+I
227 IF Q>0 THEN 230
228 LET P=INT(P/2)
229 PRINT "A HORRIBLE PLAGUE STRUCK!  HALF THE PEOPLE DIED."
230 PRINT "POPULATION IS NOW";P
232 PRINT "THE CITY NOW OWNS";A;"ACRES."
235 PRINT "YOU HARVESTED";Y;"BUSHELS PER ACRE."
250 PRINT "RATS ATE";E;"BUSHELS."
260 PRINT "YOU NOW HAVE";S;"BUSHELS IN STORE."\PRINT
270 IF Z=11 THEN 860
310 LET C=INT(10*RND(0))\LET Y=C+17
312 PRINT "LAND IS TRADING AT";Y;"BUSHELS PER ACRE."
320 PRINT "HOW MANY ACRES DO YOU WISH TO BUY";
321 INPUT Q\IF Q<0 THEN 850
322 IF Y*Q<=S THEN 330
323 GOSUB 710
324 GOTO 320
330 IF Q=0 THEN 340
331 LET A=A+Q\LET S=S-Y*Q\LET C=0
334 GOTO 400
340 PRINT "HOW MANY ACRES DO YOU WISH TO SELL";
341 INPUT Q\IF Q<0 THEN 850
342 IF Q<A THEN 350
343 GOSUB 720
344 GOTO 340
350 LET A=A-Q\LET S=S+Y*Q\LET C=0
400 PRINT
410 PRINT "HOW MANY BUSHELS DO YOU WISH TO FEED YOUR PEOPLE";
411 INPUT Q
412 IF Q<0 THEN 850
418 REM *** TRYING TO USE MORE GRAIN THAN IN THE SILOS?
420 IF Q<=S THEN 430
421 GOSUB 710
422 GOTO 410
430 LET S=S-Q\LET C=1\PRINT
440 PRINT "HOW MANY ACRES DO YOU WISH TO PLANT WITH SEED";
441 INPUT D\IF D=0 THEN 511
442 IF D<0 THEN 850
444 REM *** TRYING TO PLANT MORE ACRES THAN YOU OWN?
445 IF D<=A THEN 450
446 GOSUB 720
447 GOTO 440
```

continued on next column

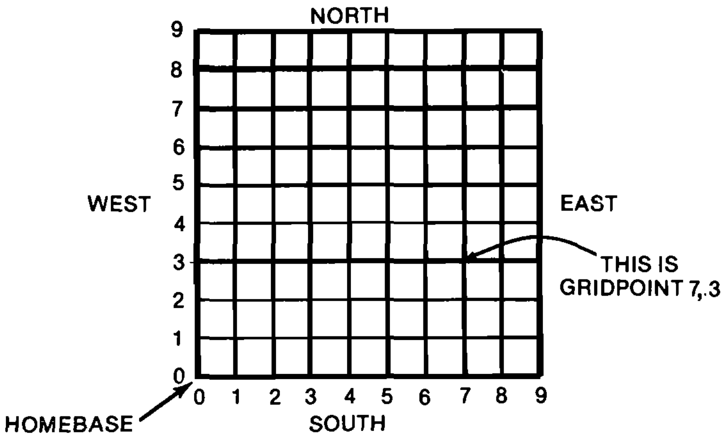
```
449 REM *** ENOUGH GRAIN FOR SEED?
450 IF INT(D/2)<S THEN 455
452 GOSUB 710
453 GOTO 440
454 REM *** ENOUGH PEOPLE TO TEND THE CROPS?
455 IF D<10*P THEN 510
460 PRINT "BUT YOU HAVE ONLY";P;"PEOPLE TO TEND THE FIELDS. NOW THEN,"
470 GOTO 440
510 LET S=S-INT(D/2)
511 GOSUB 800
512 REM *** A BOUNTYFULL HARVEST!!
515 LET Y=C\LET H=D*Y\LET E=0
521 GOSUB 800
522 IF INT(C/2)<>C/2 THEN 530
523 REM *** THE RATS ARE RUNNING WILD!!
525 LET E=INT(S/C)
530 LET S=S-E+H
531 GOSUB 800
532 REM *** LET'S HAVE SOME BABIES
533 LET I=INT(C*(20*A+S)/P/100+1)
539 REM *** HOW MANY PEOPLE HAD FULL TUMMIES?
540 LET C=INT(Q/20)
541 REM *** HORRORS, A 15% CHANCE OF PLAGUE
542 LET Q=INT(10*(2*RND(0)-.3))
550 IF P<C THEN 210
551 REM *** STARVE ENOUGH FOR IMPEACHMENT?
552 LET D=P-C\IF D>.45*P THEN 560
553 LET P1=((Z-1)*P1+D*100/P)/Z
555 LET P=C\LET D1=D1+D\GOTO 215
560 PRINT\PRINT "YOU STARVED";D;"PEOPLE IN ONE YEAR!!!"
565 PRINT "DUE TO THIS EXTREME MISMANAGEMENT YOU HAVE NOT ONLY"
566 PRINT "BEEN IMPEACHED AND THROWN OUT OF OFFICE BUT YOU HAVE"
567 PRINT "ALSO BEEN DECLARED 'NATIONAL FINK' !!"\GOTO 990
710 PRINT "HAMURABI:  THINK AGAIN. YOU HAVE ONLY"
711 PRINT S;"BUSHELS OF GRAIN.  NOW THEN,"
712 RETURN
720 PRINT "HAMURABI:  THINK AGAIN. YOU OWN ONLY";A;"ACRES.  NOW THEN,"
730 RETURN
800 LET C=INT(RND(0)*5)+1
801 RETURN
850 PRINT\PRINT "HAMURABI:  I CANNOT DO WHAT YOU WISH."
855 PRINT "GET YOURSELF ANOTHER STEWARD!!!!!"
857 GOTO 990
860 PRINT "IN YOUR 10-YEAR TERM OF OFFICE,";P1;"PERCENT OF THE"
862 PRINT "POPULATION STARVED PER YEAR ON AVERAGE, I.E., A TOTAL OF"
865 PRINT D1;"PEOPLE DIED!!"\LET L=A/P
870 PRINT "YOU STARTED WITH 10 ACRES PER PERSON AND ENDED WITH"
875 PRINT L;"ACRES PER PERSON."\PRINT
880 IF P1>33 THEN 565
885 IF L<7 THEN 565
890 IF P1>10 THEN 940
892 IF L<9 THEN 940
895 IF P1>3 THEN 960
896 IF L<10 THEN 960
900 PRINT "A FANTASTIC PERFORMANCE!!!  CHARLEMANGE, DISRAELI, AND"
905 PRINT "JEFFERSON COMBINED COULD NOT HAVE DONE BETTER!"\GOTO 990
940 PRINT "YOUR HEAVY-HANDED PERFORMANCE SMACKS OF NERO AND IVAN IV."
945 PRINT "THE PEOPLE (REMAINING) FIND YOU AN UNPLEASANT RULER, AND,"
950 PRINT "FRANKLY, HATE YOUR GUTS!"\GOTO 990
960 PRINT "YOUR PERFORMANCE COULD HAVE BEEN SOMEWHAT BETTER, BUT"
965 PRINT "REALLY WASN'T TOO BAD AT ALL. ";INT(P*.8*RND(0));"PEOPLE WOULD"
970 PRINT "DEARLY LIKE TO SEE YOU ASSASSINATED BUT WE ALL HAVE OUR"
975 PRINT "TRIVIAL PROBLEMS."
990 PRINT
995 PRINT "SO LONG FOR NOW."\PRINT
999 END
```

HURKLE

Hurkle? A Hurkle is a happy beast and lives in another galaxy on a planet named Lirht that has three moons. Hurkles are favorite pets of the Gwik, the dominant race of Lirht and... well, to find out more, read "The Hurkle is a Happy Beast" in the book *A Way Home* (by Theodore Sturgeon published by Pyramid).

In this program a shy Hurkle is hiding on a 10 by 10 grid. Homebase is point 0,0 in the **Southwest** corner. Your guess as to the gridpoint where the Hurkle is hiding should be a pair of whole numbers, separated by a comma. After each try, the computer will tell you the approximate direction to go look for the Hurkle. You get five guesses to find him.

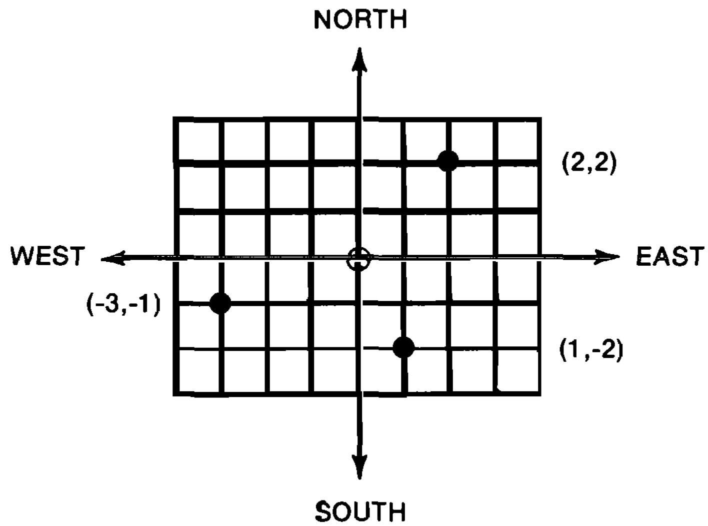
A diagram of the grid is shown on the next page.




```
LIST

HURKLE  BA    3.0     30-DEC-75

90 REM    ****    HURKLE
100 REM
101 REM
105 RANDOMIZE
110 N=5
120 G=10
210 PRINT
220 PRINT 'A HURKLE IS HIDING ON A';G;'BY';G;'GRID. HOMEBASE'
230 PRINT 'ON THE GRID IS POINT  0,0  AND ANY GRIDPOINT IS A'
240 PRINT 'PAIR OF WHOLE NUMBERS SEPARATED BY A COMMA. TRY TO'
250 PRINT 'GUESS THE HURKLE'S GRIDPOINT. YOU GET';N;'TRIES.'
260 PRINT 'AFTER EACH TRY, I WILL TELL YOU THE APPROXIMATE'
270 PRINT 'DIRECTION TO GO TO LOOK FOR THE HURKLE.'
280 PRINT
285 A=INT(G*RND(0))
286 B=INT(G*RND(0))
310 FOR K=1 TO N
320 PRINT 'GUESS #';K;
330 INPUT X,Y
340 IF ABS(X-A)+ABS(Y-B)=0 THEN 500
350 REM PRINT INFO
360 GOSUB 610
370 PRINT
380 NEXT K
410 PRINT
420 PRINT 'SORRY, THAT'S';N;'GUESSES.'
430 PRINT 'THE HURKLE IS AT ';A;',';B
440 PRINT
450 PRINT 'LET'S PLAY AGAIN. HURKLE IS HIDING.'
460 PRINT
470 GO TO 285
500 REM
510 PRINT
520 PRINT 'YOU FOUND HIM IN';K;'GUESSES!'
530 FOR I=1 TO 3 \ PRINT \ NEXT I
540 GO TO 440
610 PRINT 'GO ';
620 IF Y=B THEN 670
630 IF Y<B THEN 660
640 PRINT 'SOUTH';
650 GO TO 670
660 PRINT "NORTH';
670 IF X=A THEN 720
680 IF X<A THEN 710
690 PRINT 'WEST';
700 GO TO 720
710 PRINT 'EAST';
720 PRINT
730 RETURN
999 END
```

HURK02

This is a modified version of HURKLE. It uses positive and negative grid points and tests your ability to find the Hurkle in a "Cartesian" coordinate grid like the one below.



Points are addressed by two **coordinates,** the first corresponding to a point on the east-west axis, and the second to a point on the north-south axis. Point 0,0 is at the exact center of the grid (marked ⊕). Several other points are marked to help you find your way around.

Complete directions for playing HURK02 are contained within the program itself. More possible modifications to the program are listed below.

(1) The game could be made easier by using only Quadrant I on the Cartesian coordinate grid. This quadrant has only positive coordinates and would make the game simpler. The program might even allow the user to select the type of grid he or she wants after the instructions query.

(2) The user's distance from the Hurkle might be reported, using the formula

D = SQR(((X-H) ∧ 2) + (Y-V) ∧ 2))

This would make "HURKLE" less of a guessing game by allowing the user to calculate his or her inputs. (See "MUGWMP" in *101 BASIC Games*.)

(3) If the programmer does not wish to report the actual distance with the above formula, he or she might make the program report only whether the user is getting closer to the Hurkle or moving farther away. This would be an immense help to beginners who are not familiar with the relationships of the coordinates in each quadrant.

(4) It would also be helpful to beginners to print out the Cartesian coordinate grid either by request at the beginning or with a trace of the user's guesses if he or she does not find the Hurkle. This might be done on the copier rather than the screen. With

this feature, the user should improve much faster than with the trial and error method used in this version.

) An escape clause might be added to the guess input, allowing the user to terminate the game in the middle. This would take the form of a numeric code, for example "99999,99999". These escapes are generally good ideas to include so that the user always feels that he or she is in control of the computer rather than vice-versa.

```
LIST

HURK02  BA     3.0      30-DEC-75

1000 REM                 * * * * * * * * * * * * * * * * *
1010 REM                 *                               *
1020 REM                 *      H U R K L E    T W O     *
1030 REM                 *                               *
1040 REM                 * * * * * * * * * * * * * * * * *
1050 REM
1060 REM
1070 REM
1080 REM
1090 REM
1100 REM
1110 REM
1120 REM
1130 REM
1140 REM
 150 REM
  60 REM
  70 REM
1180 REM
1190 REM
1200 REM
1210 REM    * * * * * * *    V A R I A B L E    D I R E C T O R Y
1220 REM
1230 REM    VARIABLE        USAGE
1240 REM    --------        -----
1250 REM
1260 REM       A        INPUT CODE:  0='NO', 1='YES'
1270 REM       A$       GENERAL ALPHAMERIC USER INPUT
1280 REM       CO$      CHR$(34)  [']
1290 REM       C1$      CO$ & ', ' & CO$  [', ']
1300 REM       G1       HORIZONTAL GRID DIMENSION
1310 REM       G2       VERTICAL GRID DIMENSION
1320 REM       G3       USER-REQUESTED HORIZONTAL GRID DIMENSION
1330 REM       G4       USER-REQUESTED VERTICAL GRID DIMENSION
 340 REM       H        USER INPUT, HORIZONTAL GUESS
  50 REM       I1       TOTAL NUMBER OF GAMES PLAYED
  60 REM       I2       NUMBER OF GAMES IN WHICH THE HURKLE WAS FOUND
1370 REM       I3       NUMBER OF GUESSES IN GAMES COUNTED IN 'I2'
1380 REM       K        GENERAL FOR-NEXT LOOP INDEX
1390 REM       N        NUMBER OF TRIES ALLOWED PER HIDING PLACE
1400 REM       N1       USER-REQUESTED NUMBER OF TRIES PER HIDING PLACE
1410 REM       P0       NUMERIC ARGUMENT PASSED TO A SUBROUTINE
1420 REM       T        GUESS COUNTER
1430 REM       T$(K)    ORDINAL EXPRESSION OF GUESS NUMBER
1440 REM       V        USER INPUT, VERTICAL GUESS
1450 REM       X        HORIZONTAL COORDINATE OF HURKLE'S HIDING PLACE
1460 REM       Y        VERTICAL COORDINATE OF HURKLE'S HIDING PLACE
1470 REM
1480 REM
1490 REM    * * * * * * *    D E C L A R A T I O N S
1500 REM
1510 LET CO$=CHR$(34)
1520 LET C1$=CO$ & ', ' & CO$
 530 DIM T$(10,8)
  40 REM -- FOR DECSYSTEM 10, REPLACE ABOVE STATEMENT WITH:
  50 REM          DIM T$(10)
1560 DATA 'FIRST', 'SECOND', 'THIRD', 'FOURTH', 'FIFTH'
1570 DATA 'SIXTH', 'SEVENTH', 'EIGHTH', 'NINTH', 'TENTH'
1580 FOR K=1 TO 10
1590 READ T$(K)
1600 NEXT K
1610 LET I1=0
1620 LET I2=0
1630 REM
1640 REM
1650 REM                     ------------------------
1660 REM    * * * * * * *    M A I N    P R O G R A M
1670 REM                     ------------------------
1680 REM
1690 REM
1700 GOSUB 3820
1710 PRINT 'HURKLE TWO'
1720 PRINT '----------'
1730 PRINT
1740 PRINT
1750 PRINT 'DO YOU WISH TO SEE THE INSTRUCTIONS ('; CO$; 'YES'; CO$;
1760 PRINT ' OR '; CO$; 'NO'; CO$; ')';
1770 RANDOMIZE
1780 LET G1=6+2*INT(4*RND(0))
1790 LET G2=6+2*INT(4*RND(0))
1800 LET N=5+INT(3*RND(0))
1810 GOSUB 3870
1820 IF A=0 THEN 1870
1830 GOSUB 4170
1840 REM
1850 REM    * * * * * * *    O P T I O N    I N P U T
 360 REM
  70 PRINT
 380 PRINT 'YOUR AVAILABLE OPTIONS ARE NOW ';
1890 P0=1
```

continued on next column

```
1900 GOSUB 4060
1910 PRINT 'WHICH WOULD YOU LIKE TO EXERCISE (ENTER A WORD)';
1920 INPUT A$
1930 PRINT
1940 IF A$='GO' THEN 2090
1950 IF A$='HELP' THEN 2960
1960 IF A$='INSTR' THEN 1830
1970 IF A$='QUIT' THEN 4540
1980 IF A$='SIZE' THEN 3140
1990 IF A$='TRIES' THEN 3570
2000 PRINT 'PLEASE ENTER ONLY ';
2010 P0=2
2020 GOSUB 4060
2030 PRINT '('; CO$; 'HELP'; CO$; ' PRINTS AN EXPLANATION OF EACH.)  ';
2040 PRINT 'YOUR CHOICE';
2050 GOTO 1920
2060 REM
2070 REM    * * * * * * *    T H E    ' G O '    O P T I O N
2080 REM
2090 REM
2100 REM  *******  SET THE HURKLE'S COORDINATES
2110 REM
2120 X=-G1/2+INT((G1*1)*RND(0))
2130 Y=-G2/2+INT((G2+1)*RND(0))
2140 PRINT C2$;
2150 GOSUB 3820
2160 PRINT 'THE HURKLE IS HIDING IN A';
2170 IF SEG$(STR$(G1),LEN(STR$(G1)),LEN(STR$(G1)))<>'8' THEN 2210
2180 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
2190 REM          IF RIGHT$(STR$(G1),LEN(STR$(G1)))<>'8' THEN 2210
2200 PRINT 'N';
2210 PRINT G1; 'BY'; G2; 'COORDINATE GRID.  HORIZONTAL'
2220 PRINT 'VALUES GO FROM '; G1/-2; 'TO'; G1/2; 'AND VERTICAL ';
2230 PRINT 'VALUES GO FROM '; G2/-2; 'TO'; G2/2; '.  FIND'
2240 PRINT 'THE HURKLE WITHIN'; N; 'GUESSES!'
2250 PRINT
2260 REM
2270 REM  *******  INPUT THE GUESSES
2280 REM
2290 FOR T=1 TO N
2300 PRINT
2310 IF T>10 THEN 2340
2320 PRINT 'YOUR '; T$(T);
2330 GOTO 2350
2340 PRINT 'YOUR'; T; 'TH';
2350 PRINT ' GUESS';
2360 IF T>1 THEN 2380
2370 PRINT ' (ENTER COORDINATES SEPARATED BY A COMMA)';
2380 INPUT H,V
2390 REM
2400 REM  *******  CHECK GUESSES FOR VALIDITY
2410 REM
2420 IF H<-G1/2 THEN 2470
2430 IF H>+G1/2 THEN 2470
2440 IF V<-G2/2 THEN 2490
2450 IF V>+G2/2 THEN 2490
2460 GOTO 2550
2470 PRINT '   YOUR FIRST';
2480 GOTO 2500
2490 PRINT '   YOUR SECOND';
2500 PRINT ' COORDINATE IS OUTSIDE OF THE HURKLE'S GRID!  TRY AGAIN...'
2510 GOTO 2300
2520 REM
2530 REM  *******  EVALUATE A VALID GUESS
2540 REM
2550 IF ABS(X-H)+ABS(Y-V)=0 THEN 2720
2560 IF N=T THEN 2620
2570 GOSUB 4380
2580 NEXT T
2590 REM
2600 REM  *******  OUT OF GUESSES
2610 REM
2620 PRINT
2630 PRINT
2640 PRINT 'SORRY, BUT YOU HAVE HAD THE LIMIT OF'; N; 'GUESSES.  THE ';
2650 PRINT 'HURKLE WAS HIDING'
2660 PRINT 'AT POINT  '; STR$(X); ','; STR$(Y); ' .'
2670 P0=0
2680 GOTO 2810
2690 REM
2700 REM  *******  FOUND HURKLE MESSAGE
2710 REM
2720 PRINT
2730 IF T>5 THEN 2790
2740 FOR K=1 TO 6-T
2750 PRINT 'HURK!  '; PNT(7);
2760 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
2770 REM          PRINT 'HURK!  '; CHR$(7);
2780 NEXT K
2790 PRINT 'YOU FOUND THE HURKLE IN'; T; 'GUESSES!!'
2800 P0=1
2810 PRINT
2820 PRINT 'IF YOU'D LIKE TO PLAY AGAIN, PLEASE ENTER THE '; CO$; 'GO';
2830 PRINT CO$; ' OPTION BELOW.'
2840 PRINT
2850 REM
2860 REM  *******  INCREMENT THE GAME AND TOTAL GUESSES COUNTERS
2870 REM
2880 LET I1=I1+1
2890 LET I2=I2+P0
2900 IF P0=0 THEN 1870
2910 LET I3=I3+T
2920 GOTO 1870
2930 REM
2940 REM    * * * * * * *    T H E    ' H E L P '    O P T I O N
2950 REM
2960 GOSUB 3820
2970 PRINT 'YOUR OPTIONS PERFORM THE FOLLOWING FUNCTIONS:'
2980 PRINT '   GO       LOCATE THE HURKLE AT A NEW GRID POINT AND ';
2990 PRINT 'ALLOW YOU TO'
3000 PRINT '            GUESS WHERE IT IS HIDING'
3010 PRINT '   HELP     DISPLAY THIS MESSAGE'
3020 PRINT '   INSTR    DISPLAY THE INSTRUCTIONS'
3030 PRINT '   QUIT     END THE GAME'
3040 PRINT '   SIZE     CHANGE THE SIZE OF THE GRID IN WHICH THE ';
```

continued on next page

```
3050 PRINT "HURKLE CAN HIDE"
3060 PRINT "   TRIES   CHANGE THE NUMBER OF TRIES ALLOWED TO FIND ";
3070 PRINT "THE HURKLE"
3080 PRINT "TO MAKE THE COMPUTER EXERCISE AN OPTION, SIMPLY TYPE ";
3090 PRINT "ITS KEYWORD BELOW."
3100 GOTO 1870
3110 REM
3120 REM    * * * * * * *    T H E    " S I Z E "    O P T I O N
3130 REM
3140 GOSUB 3820
3150 PRINT "THE CURRENT SIZE OF THE HURKLE'S GRID IS"; G1; "BY"; G2;
3160 PRINT "(HORIZONTAL BY"
3170 PRINT "VERTICAL).  YOUR NEW DIMENSIONS MUST ALSO BE EVEN ";
3180 PRINT "INTEGERS.  ENTER YOUR"
3190 PRINT "NEW DIMENSIONS BELOW SEPARATED BY A COMMA, HORIZONTAL ";
3200 PRINT "DIMENSION FOLLOWED"
3210 PRINT "BY VERTICAL.  YOU MAY LEAVE THE GRID SIZE UNCHANGED BY ";
3220 PRINT "ENTERING "; CO$; "0,0"; CO$; "."
3230 PRINT
3240 PRINT "YOUR NEW DIMENSIONS";
3250 INPUT G3,G4
3260 IF G3<>0 THEN 3280
3270 IF G4=0 THEN 3410
3280 IF G3/2<>INT(G3/2) THEN 3490
3290 IF G4/2<>INT(G4/2) THEN 3510
3300 REM
3310 REM  *******  VALID INPUT
3320 REM
3330 LET G1=G3
3340 LET G2=G4
3350 PRINT
3360 PRINT "THE NEW SIZE OF THE HURKLE'S GRID IS"; G1; "BY"; G2; "."
3370 GOTO 3440
3380 REM
3390 REM  *******  0,0 INPUT
3400 REM
3410 PRINT
3420 PRINT "THE HURKLE'S GRID WILL REMAIN ITS CURRENT SIZE OF"; G1;
3430 PRINT "BY"; G2; "."
3440 PRINT
3450 GOTO 1870
3460 REM
3470 REM  *******  NON-INTEGER INPUT
3480 REM
3490 PRINT "   YOUR FIRST";
3500 GOTO 3520
3510 PRINT "   YOUR SECOND";
3520 PRINT " DIMENSION IS NOT AN EVEN INTEGER!  PLEASE TRY AGAIN..."
3530 GOTO 3230
3540 REM
3550 REM    * * * * * * *    T H E    " T R I E S "    O P T I O N
3560 REM
3570 GOSUB 3820
3580 PRINT "YOU ARE NOW ALLOWED"; N; "TRIES TO FIND THE HURKLE.  ";
3590 PRINT "ENTER YOUR NEW"
3600 PRINT "LIMIT BELOW.  YOU MAY LEAVE THE LIMIT UNCHANGED BY ";
3610 PRINT "ENTERING "; CO$; "0"; CO$; "."
3620 PRINT
3630 PRINT "YOUR NEW LIMIT";
3640 INPUT N1
3650 PRINT
3660 IF N1<>0 THEN 3690
3670 PRINT "THE NUMBER OF TRIES ALLOWED WILL REMAIN AT"; N; "."
3680 GOTO 3710
3690 LET N=N1
3700 PRINT "YOU WILL NOW BE ALLOWED"; N; "TRIES TO FIND THE HURKLE."
3710 PRINT
3720 GOTO 1870
3730 REM
3740 REM
3750 REM                       ---------------------
3760 REM    * * * * * * *    S U B R O U T I N E S
3770 REM                       ---------------------
3780 REM
3790 REM
3800 REM  *******  SCREEN CLEARER
3810 REM
3820 PRINT PNT(27); "H"; PNT(27); "J";
3830 RETURN
3840 REM
3850 REM  *******  "YES", "NO", AND "QUIT" RESPONSE DECODER
3860 REM
3870 INPUT A$
3880 PRINT
3890 IF POS(A$,"Y",1)<>0 THEN 3990
3900 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
3910 REM          IF INSTR(1,A$,"Y")<>0 THEN 3990
3920 IF POS(A$,"N",1)<>0 THEN 4010
3930 REM -- FOR DECSYSTEM-10, REPLACE ABOVE STATEMENT WITH:
3940 REM          IF INSTR(1,A$,"N")<>0 THEN 4010
3950 IF A$="QUIT" THEN 4540
3960 PRINT "     PLEASE INPUT "; CO$; "YES"; C1$; "NO"; CO$; ", OR ";
3970 PRINT CO$; "QUIT"; CO$; ".  YOUR CHOICE";
3980 GOTO 3870
3990 LET A=1
4000 RETURN
4010 LET A=0
4020 RETURN
4030 REM
4040 REM    * * * * * * *    O P T I O N    P R I N T E R
4050 REM
4060 PRINT CO$; "GO"; C1$; "HELP"; C1$; "INSTR"; C1$; "QUIT"; C1$;
4070 PRINT "SIZE"; CO$;
4080 IF PO=2 THEN 4120
4090 PRINT ","
4100 PRINT "AND "; CO$; "TRIES"; CO$; ".  ";
4110 RETURN
4120 PRINT ", OR "; CO$; "TRIES"; CO$; "."
4130 RETURN
4140 REM
4150 REM    * * * * * * *    I N S T R U C T I O N S
4160 REM
4170 GOSUB 3820
4180 PRINT "A HURKLE IS HIDING IN A CARTESIAN COORDINATE GRID"
```

continued on next column

```
4190 PRINT "LIKE THE ONE AT THE RIGHT.  GUESS ITS LOCATION BY";
4200 PRINT "              (POS)"
4210 PRINT "ENTERING A HORIZONTAL COORDINATE FOLLOWED BY A   ";
4220 PRINT "                N"
4230 PRINT "VERTICAL ONE.  FOR EXAMPLE, THE  *  IS AT  -4,1 .";
4240 PRINT "           *   !"
4250 PRINT "POINT  0,0  IS AT THE CENTER OF THE GRID, WHERE  ";
4260 PRINT "         W  <--+-->  E"
4270 PRINT "THE  +  IS.  AFTER EACH GUESS, I WILL TELL YOU   ";
4280 PRINT "       (NEG)   !   (POS)"
4290 PRINT "WHERE TO GO TO FIND THE HURKLE BY SAYING "; CO$; "NORTH";
4300 PRINT CO$; "                S"
4310 PRINT "FOR THE POSITIVE VERTICAL DIRECTION, "; CO$; "WEST"; CO$;
4320 PRINT " FOR                (NEG)"
4330 PRINT "THE NEGATIVE HORIZONTAL, ETC.  GOOD LUCK!"
4340 RETURN
4350 REM
4360 REM    * * * * * * *    D I R E C T I O N A L    H I N T E R
4370 REM
4380 PRINT "GO ";
4390 IF V=Y THEN 4440
4400 IF V>Y THEN 4430
4410 PRINT "NORTH";
4420 GOTO 4440
4430 PRINT "SOUTH";
4440 IF H=X THEN 4490
4450 IF H>X THEN 4480
4460 PRINT "EAST";
4470 GOTO 4490
4480 PRINT "WEST";
4490 PRINT "..."
4500 RETURN
4510 REM
4520 REM    * * * * * * *    Q U I T T I N G    R O U T I N E
4530 REM
4540 PRINT
4550 IF I1<2 THEN 4650
4560 PRINT "YOU PLAYED A TOTAL OF"; I1; "GAMES AND FOUND THE ";
4570 PRINT "HURKLE IN"; I2; "OF THEM."
4580 PRINT "THAT'S A WINNING PERCENTAGE OF"; 100*I2/I1; "% !"
4590 PRINT
4600 IF I2<2 THEN 4650
4610 PRINT "IN THE"; I2; "GAMES YOU WON, IT TOOK YOU AN AVERAGE OF";
4620 PRINT I3/I2; "GUESSES TO"
4630 PRINT "FIND THE HURKLE."
4640 PRINT
4650 PRINT "BYE!"
4660 PRINT
4670 END
```

MORGAG

Program name: MORGAG.BA

Purpose: Given:

(1) an amount of money borrowed,
(2) the yearly interest rate to be paid, and
(3) the number of years allowed to pay back the loan,

the program will calculate:

(1) the monthly interest rate,
(2) the number of months allowed to pay back the loan, and
(3) the amount of money to be paid each month.

In addition, it prints the following data for each month:

(1) the number of the payment,
(2) the amount of money still unpaid (the outstanding principal),
(3) the amount of interest paid that month,
(4) the amount of principal paid that month,
(5) the total amount of interest paid to date, and
(6) the total amount of principal paid to date.

How it works. The data to be given to this program (as mentioned above) are supplied through the INPUT statements at lines 1780, 1820, and 1860.

The amount to be paid each month is calculated with the formula

$$M = \frac{P*I}{1 - \frac{1}{(1+I)^T}}$$

where M is the monthly payment
P is the amount of the loan
I is the monthly interest rate

T is the number of months allowed to pay back the loan

and these values are displayed on the screen.

MORGAG then asks whether or not the user wishes to see a monthly breakdown of the mortgage payments. This section prints the following values for each monthly payment:

(1) monthly payment number,
(2) outstanding principal,
(3) interest payment,
(4) principal payment,
(5) total interest paid to date, and
(6) total principal paid to date.

This table may be printed on the screen or in a disk file called "MORT.MO" on RXA1: or it may be omitted. The action to be taken is determined by the response to the query at line 2440.

Limitations. This program takes quite a long time to run. Therefore, it is recommended that the monthly breakdown be directed to a disk file rather than the screen. This data can then be printed on the screen or copier with the monitor TYPE command.

The program is not totally accurate for loans of \$10000.00 or more because CLASSIC BASIC can only store numbers to an accuracy of 6 digits. That is, some pennies will be lost as \$29078.33 will be stored as \$29078.3. However, the cumulative error in a \$30000 mortgage was found to be less than \$3.00 at a yearly interest rate of 9.5%, or less than 0.01% error.

Listing. A complete program listing is shown below.

```
LIST

MORGAG  BA    3.0     30-DEC-75

1000 REM                -----------
1020 REM    ****    M O R G A G
1040 REM                -----------
1060 REM
1080 REM
1100 REM
1120 REM
1140 REM
1160 REM    ****    VARIABLE DIRECTORY
1180 REM
1200 REM    VARIABLE        USAGE
1220 REM    --------        ------
1240 REM        I       MONTHLY INTEREST
1260 REM        IO      INTEREST PAYMENT
1280 REM        I2      TOTAL INTEREST PAID
1300 REM        K       MONTH
1320 REM        M       MONTHLY PAYMENT (PRINCIPAL + INTEREST)
1340 REM        P       PRINCIPAL
1360 REM        P2      TOTAL PRINCIPAL PAID
1380 REM        Q       YEARLY INTEREST RATE
1400 REM        T       TERM
1420 REM        Y$      OUTPUT MEDIUM
1440 REM
1460 REM    ****    DECLARATIONS
1480 REM
1500 DEF FNR(X)=INT((100*X)+.5)/100
1520 DEF FNS(X)=LEN(STR$(INT(X)))
1540 REM
1560 REM                ------------------------
1580 REM    ****    M A I N    P R O G R A M
1600 REM                ------------------------
1620 REM
1660 PRINT 'COMPUTATION OF MORGAGE PAYMENTS'
1680 PRINT
1720 REM    ****    PRINCIPAL, INTEREST, AND TERM QUERIES
1740 REM
1760 PRINT 'PLEASE INPUT THE PRINCIPAL (WITHOUT COMMAS)';
1780 INPUT P
1800 PRINT 'INPUT THE ANNUAL INTEREST RATE (IN %)';
1820 INPUT I
1840 PRINT 'INPUT THE TERM (IN YEARS)';
1860 INPUT T
1880 PRINT
1920 REM    ****    CONVERT TO MONTHLY FIGURES
1940 REM
1960 LET T=T*12
1980 LET Q=I
2000 LET I=I/1200
2020 REM
2040 REM    ****    COMPUTE MONTHLY PAYMENT
```

continued on next column

```
2060 REM
2080 LET M=FNR(P*I/(1-1/(1+I)^T))
2100 REM
2120 REM    ****    PRINT SUMMARY REPORT
2140 REM
2160 FILEV#1:'TTY:'
2200 PRINT 'PRINCIPAL'; TAB(30); '$'; P
2220 PRINT 'INTEREST RATE'; TAB(35); Q; '%'
2240 PRINT 'TERM'; TAB(33); T; TAB(40); 'MONTHS'
2260 PRINT 'MONTHLY PAYMENT'; TAB(30); '$'; TAB(36-FNS(M)); M
2270 PRINT
2280 REM
2300 REM    ****    MONTHLY BREAKDOWN QUERY
2320 REM
2340 PRINT 'IF YOU WANT THE MONTHLY BREAKDOWN ON THE SCREEN,';
2360 PRINT ' ENTER ''SCREEN''.'
2380 PRINT 'IF YOU WANT IT ON DISK ENTER ''DISK''.';
2400 PRINT 'IF YOU DON'T WANT IT AT ALL ENTER ''NO''.'
2420 PRINT 'YOUR ENTRY';
2440 INPUT Y$
2460 IF Y$='NO' GOTO 3420
2480 IF Y$='DISK' GOTO 2560
2500 IF Y$='SCREEN' GOTO 2640
2520 PRINT '  PLEASE ENTER ''SCREEN'', ''DISK'', OR ''NO''.  ';
2540 GOTO 2420
2560 CLOSE #1
2580 FILEV#1:'RXA1:MORT.MO'
2600 PRINT
2620 PRINT
2640 PRINT #1:
2660 PRINT#1:
2680 REM
2700 REM    ****    PRINT HEADING LINES
2720 REM
2740 PRINT #1: TAB(8); 'OUTSTANDING'; TAB(23); 'INTEREST'; TAB(35);
2760 PRINT #1: 'PRINCIPAL';TAB(50);'TOTAL';TAB(64);'TOTAL'
2780 PRINT #1: 'MONTH';TAB(9);'PRINCIPAL';TAB(24);'PAYMENT';
2800 PRINT #1: TAB(36); 'PAYMENT'; TAB(48); 'INTEREST'; TAB(62);
2820 PRINT #1: 'PRINCIPAL'
2840 PRINT #1:
2860 FOR K=1 TO T
2880 REM
2900 REM    ****    COMPUTE MONTHLY PAYMENT BREAKDOWN
2920 REM
2940 LET IO=FNR(P*I)
2960 LET P2=FNR(P2+M-IO)
2980 LET I2=FNR(I2+IO)
3000 REM
3020 REM    ****    PRINT MONTHLY PAYMENT BREAKDOWN
3040 REM
3060 PRINT #1: TAB(4-FNS(K)); STR$(K);
3080 PRINT #1:  TAB(14-FNS(P)); P;
3100 PRINT #1:  TAB(26-FNS(IO)); IO;
3120 PRINT #1:  TAB(38-FNS(M-IO)); M-IO;
3140 PRINT #1:  TAB(52-FNS(I2)); I2;
3160 PRINT #1:  TAB(66-FNS(P2)); P2
3180 REM
3200 REM    ****    COMPUTE OUTSTANDING PRINCIPAL
3220 REM
3240 LET P=FNR(P-(M-IO))
3260 REM
3280 REM    ****    COUNT MONTHS ON DISK
3300 REM
3320 IF Y$='SCREEN' GOTO 3400
3340 IF K/12=INT(K/12) GOTO 3380
3360 GOTO 3400
3380 PRINT 'JUST FINISHED MONTH #';K
3400 NEXT K
3420 CLOSE #1
3440 END
```

QUADEQ, QUAD02, and QUAD03

QUADEQ is a program similar to EASY02, but it finds the roots of quadratic equations (see page 5-29). Although short and to the point, QUADEQ can differentiate between equations with real and complex roots and solve either type.

QUAD02 and QUAD03 are adaptions of QUADEQ, made by adding REMark statements to clarify the program through documentation. The executable statements in these three programs are all exactly the same. See Module 5-C for a further discussion of the documentation of these programs.

```
LIST

QUADEQ  BA    3.0     30-DEC-75

10 PRINT 'THIS PROGRAM WILL SOLVE THE QUADRATIC EQUATION IN THE FORM:'
20 PRINT 'AX^2 + BX + C = 0.'
30 PRINT 'AFTER EACH ?, TYPE THE REQUESTED VALUE & PUSH RETURN.'
40 PRINT\PRINT 'A = ';\INPUT A
50 PRINT'B = ';\INPUT B\PRINT  'C = ';\INPUT C
60 D=B^2 - 4*A*C
70 IF D < 0 THEN 110
80 R1=(-B+SQR(D))/(2*A) \ R2=(-B-SQR(D))/(2*A)
90 PRINT 'THE ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
100 PRINT R1 \ PRINT R2 \ GOTO 140
110 P1=-B/(2*A) \ P2=SQR(ABS(D))/(2*A)
120 PRINT 'THE COMPLEX ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
130 PRINT P1; ' +'; P2; 'I' \ PRINT P1; ' -'; P2; 'I'
140 PRINT\PRINT 'DO YOU WISH TO SOLVE ANOTHER QUADRATIC EQUATION?'
150 PRINT 'ANSWER YES OR NO & PUSH RETURN.'; \ INPUT Q$
160 IF Q$='YES' THEN 40
170 END
```

continued on next page

```
LIST

QUAD02  BA    3.0      30-DEC-75

100 REM     ****     Q U A D 0 2
110 REM
120 REM
130 REM     ****     DIRECTIONS
140 REM
150 PRINT 'THIS PROGRAM WILL SOLVE THE QUADRATIC EQUATION IN THE FORM:'
160 PRINT 'AX^2 + BX + C = 0.'
170 PRINT 'AFTER EACH ?, TYPE THE REQUESTED VALUE & PUSH RETURN.'
180 REM
190 REM     ****     INPUT OF A, B, AND C
200 REM
210 PRINT\PRINT 'A = ';\INPUT A
220 PRINT'B = ';\INPUT B\PRINT  'C = ';\INPUT C
230 REM
240 REM     ****     CALCULATION OF THE DETERMINANT
250 REM
260 D=B^2 - 4*A*C
270 IF D < 0 THEN 370
280 REM
290 REM     ****     CALCULATION OF REAL ROOTS
300 REM
310 R1=(-B+SQR(D))/(2*A) \ R2=(-B-SQR(D))/(2*A)
320 PRINT 'THE ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
330 PRINT R1 \ PRINT R2 \ GOTO 430
340 REM
350 REM     ****     CALCULATION OF COMPLEX ROOTS
360 REM
370 P1=-B/(2*A) \ P2=SQR(ABS(D))/(2*A)
380 PRINT 'THE COMPLEX ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
390 PRINT P1; ' +'; P2; 'I' \ PRINT P1; ' -'; P2; 'I'
400 REM
410 REM     ****     RERUN QUERY
420 REM
430 PRINT\PRINT 'DO YOU WISH TO SOLVE ANOTHER QUADRATIC EQUATION?'
440 PRINT 'ANSWER YES OR NO & PUSH RETURN.'; \ INPUT Q$
450 IF Q$='YES' THEN 210
460 END
```

```
LIST

QUAD03  BA    3.0      30-DEC-75

100 REM     ****     Q U A D 0 3
110 REM
120 REM
130 REM
140 REM
150 REM
160 REM
170 REM     ****     VARIABLE DIRECTORY
180 REM
190 REM     VARIABLE        USAGE
200 REM     --------        -----
210 REM        A        USER INPUT, COEFFICIENT OF 'X^2' TERM
220 REM        B        USER INPUT, COEFFICIENT OF 'X' TERM
230 REM        C        USER INPUT, 'C' TERM
240 REM        D        DISCRIMINANT
250 REM        P1       REAL PART OF A COMPLEX ROOT
260 REM        P2       IMAGINERY PART OF A COMPLEX ROOT
270 REM        Q$       USER RESPONSE TO RERUN QUERY
280 REM        R1       FIRST REAL ROOT
290 REM        R2       SECOND REAL ROOT
300 REM
310 REM     ****     DIRECTIONS
320 REM
330 PRINT 'THIS PROGRAM WILL SOLVE THE QUADRATIC EQUATION IN THE FORM:'
340 PRINT 'AX^2 + BX + C = 0.'
350 PRINT 'AFTER EACH ?, TYPE THE REQUESTED VALUE & PUSH RETURN.'
360 REM
370 REM     ****     INPUT OF A, B, AND C
380 REM
390 PRINT\PRINT 'A = ';\INPUT A
400 PRINT'B = ';\INPUT B\PRINT  'C = ';\INPUT C
410 REM
420 REM     ****     CALCULATION OF THE DETERMINANT
430 REM
440 D=B^2 - 4*A*C
450 IF D < 0 THEN 550
460 REM
470 REM     ****     CALCULATION OF REAL ROOTS
480 REM
490 R1=(-B+SQR(D))/(2*A) \ R2=(-B-SQR(D))/(2*A)
500 PRINT 'THE ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
510 PRINT R1 \ PRINT R2 \ GOTO 610
520 REM
530 REM     ****     CALCULATION OF COMPLEX ROOTS
540 REM
550 P1=-B/(2*A) \ P2=SQR(ABS(D))/(2*A)
560 PRINT 'THE COMPLEX ROOTS OF '; A; 'X^2 +'; B; 'X +'; C; ' = 0 ARE:'
570 PRINT P1; ' +'; P2; 'I' \ PRINT P1; ' -'; P2; 'I'
580 REM
590 REM     ****     RERUN QUERY
600 REM
610 PRINT\PRINT 'DO YOU WISH TO SOLVE ANOTHER QUADRATIC EQUATION?'
620 PRINT 'ANSWER YES OR NO & PUSH RETURN.'; \ INPUT Q$
630 IF Q$='YES' THEN 390
640 END
```

SYNONY and SYNSET

SYNONY is a CAI application program that can be used in both drill-and-practice and testing modes. The program presents the student with a word and asks him or her to supply a synonym. As a drill, students can run this program over and over, as each word presented accepts at least four different correct answers as synonyms. As a test, a single run of SYNONY could evaluate a specific lesson on synonyms.

This program maintains a data file called "SYCOR.TS" on RXA1: which stores the total number of times that a correct synonym was entered for each word. This file must be created by the program RXA1:SYNSET before SYNONY can be used. An EN error message is printed by SYNONY if "SYCOR.TS" does not exist. The problem is corrected simply by running SYNSET.

Many other words and correct answers could be added to SYNONY by supplying additional DATA statements. The number "10" in the DATA statement at line 3740 tells the program how many different sets of words follow. The numbers in subsequent data statements indicate the number of synonyms in that set. These numbers must be adjusted when additional data are entered.

```
LIST

SYNONY  BA    3.0      30-DEC-75

1000 REM              -----------
1020 REM     ****     S Y N O N Y
1040 REM              -----------
1060 REM
1080 REM
1100 REM
1120 REM
1140 REM
1160 REM
1180 REM
1200 REM
1220 REM     ****     VARIABLE DIRECTORY
1240 REM
1260 REM     VARIABLE        USAGE
1280 REM     --------        -----
1300 REM        A$       STUDENT'S ANSWER
1320 REM        C        NUMBER OF CURRENT QUESTION
1340 REM        J        POINTER TO WORD IN LIST OF SYNONYMS
1360 REM        N        TOTAL NUMBER OF QUESTIONS
1380 REM        N2       NUMBER OF SYNONYMS FOR CURRENT WORD
1400 REM        Q$       FILE NAME OF SCORE FILE
1420 REM        R$       ANSWER TO TOTAL SCORES QUERY
1440 REM        S        SCORE ARRAY ROW POINTER
1460 REM        T        SCORE ARRAY COLUMN POINTER
1480 REM        V$       LIST OF WORDS USED IN QUESTIONS
1500 REM        W$       LIST OF SYNONYMS FOR CURRENT WORD
1520 REM        X        ARRAY OF TOTAL SCORES
1540 REM
1560 REM     ****     DECLARATIONS
1580 REM
1600 DIM A$(70)
1620 DIM Q$(70)
1640 DIM X(2,10),W$(10,20)
1660 DEF FNS(X)=LEN(STR$(INT(X)))
1680 DIM V$(10,20)
1700 REM              ------------------------
1720 REM     ****     M A I N    P R O G R A M
1740 REM              ------------------------
1760 REM
1780 REM     ****     PRINT SYNSET MESSAGE
1800 REM
1820 PRINT 'SYNONYMS'
1840 PRINT
1860 LET Q$='RXA1:SYCOR.TS'
1880 PRINT 'IF YOU SEE THE MESSAGE:  EN AT LINE 2020'
1900 PRINT 'BELOW, RUN THE PROGRAM ''SYNSET'' BY TYPING:'
1920 PRINT '        OLD RXA1:SYNSET'
1940 PRINT 'AND THEN:'
1960 PRINT '        RUN'
1980 PRINT
2000 PRINT 'MESSAGE:'
2020 FILEN#1:Q$
2040 PRINT 'NO ERROR MESSAGE'
2060 REM
2080 REM     ****     READ FILE OF SCORES
2100 REM
2120 FOR S=1 TO 2
2140 FOR T= 1 TO 10
2160 INPUT #1: X(S,T)
2180 NEXT T
2200 NEXT S
2220 CLOSE #1
2240 PRINT
2260 REM
2280 REM     ****     PRINT EXPLANATION FOR USER
2300 REM
2320 PRINT 'A SYNONYM OF A WORD IS ANOTHER WORD IN THE ENGLISH LANGUAGE'
2340 PRINT 'WHICH HAS THE SAME OR VERY NEARLY THE SAME MEANING.'
2360 PRINT
2380 PRINT 'I CHOOSE A WORD -- YOU TYPE A SYNONYM.'
2400 REM
2420 REM     ****     READ # OF QUESTIONS & COUNT THEM
2440 REM
2460 READ N
2480 LET C =C+1
```

continued on next page

```
2500 PRINT
2520 IF C>N THEN 3120
2540 REM
2560 REM     ****     READ A LINE OF SYNONYMS
2580 REM
2600 READ N2
2620 FOR J=1 TO N2
2640 READ W$(J)
2660 NEXT J
2680 REM
2700 REM     ****     STORE CURRENT WORD & ASK QUESTION
2720 REM
2740 LET V$(C)=W$(1)
2760 PRINT 'WHAT IS A SYNONYM OF ';W$(1);
2780 INPUT A$
2800 REM
2820 REM     ****     TEST TO SEE IF THE ANSWER IS CORRECT
2840 REM     ****     AND ADD TO THE SCORE FOR THAT WORD
2860 REM
2880 FOR J=2 TO N2
2900 IF A$=W$(J) THEN 3000
2920 NEXT J
2940 PRINT '  WRONG'
2960 LET X(2,C)=X(2,C)+1
2980 GOTO 2480
3000 PRINT '  CORRECT'
3020 LET X(1,C)=X(1,C)+1
3040 GOTO 2480
3060 REM
3080 REM     ****     ASK IF USER WANTS TO SEE TOTAL SCORES
3100 REM
3120 PRINT 'SYNONYM DRILL COMPLETED.'
3140 PRINT
3160 PRINT 'DO YOU WANT TO SEE THE TOTAL SCORES?'
3180 PRINT 'ANSWER ''YES'' OR ''NO''.  YOUR CHOICE';
3200 INPUT R$
3220 IF R$='YES' GOTO 3580
3240 IF R$='NO' GOTO 3380
3260 PRINT
3280 PRINT 'PLEASE ';
3300 GOTO 3180
3320 REM
3340 REM     ****     OUTPUT TOTAL SCORES FILE
3360 REM
3380 FILEVN#1:Q$
3400 FOR S=1 TO 2
3420 FOR T=1 TO 10
3440 PRINT #1: X(S,T)
3460 NEXT T
3480 NEXT S
3500 GOTO 3800
3520 REM
3540 REM     ****     DISPLAY TOTAL SCORES
3560 REM
3580 PRINT
3600 PRINT TAB(14);'WORD        TIMES      TIMES'
3620 PRINT TAB(14);'            CORRECT    WRONG'
3640 PRINT
3660 FOR T=1 TO N
3680 PRINT TAB(18-LEN(V$(T)));V$(T);
3700 PRINT TAB(27-FNS(X(1,T)));X(1,T);
3720 PRINT TAB(37-FNS(X(2,T)));X(2,T)
3740 NEXT T
3760 PRINT
3780 GOTO 3380
3800 CLOSE #1
3820 REM
3840 REM     ****     NUMBER OF QUESTIONS & LISTS OF SYNONYMS
3860 REM
3880 DATA 10
3900 DATA 5,'FIRST','START','BEGINNING','ONSET','INITIAL'
3920 DATA 5,'SIMILAR','ALIKE','SAME','LIKE','RESEMBLING'
3940 DATA 5,'MODEL','PATTERN','PROTOTYPE','STANDARD','CRITERION'
3960 DATA 5,'SMALL','INSIGNIFICANT','LITTLE','TINY','MINUTE'
3980 DATA 6,'STOP','HALT','STAY','ARREST','CHECK','STANDSTILL'
4000 DATA 6, 'HOUSE','DWELLING','RESIDENCE','DOMICILE','LODGING'
4020 DATA 'HABITATION'
4040 DATA 7,'PIT','HOLLOW','HOLE','WELL','GULF','CHASM','ABYSS'
4060 DATA 7,'PUSH','SHOVE','THRUST','PROD','POKE','BUTT','PRESS'
4080 DATA 6,'RED','ROUGE','SCARLET','CRIMSON','FLAME','RUBY'
4100 DATA 7,'PAIN','SUFFERING','HURT','MISERY','DISTRESS','ACHE'
4120 DATA 'DISCOMFORT'
4140 END
```

```
LIST

SYNSET  BA    3.0     30-DEC-75

1000 REM              -----------
1020 REM     ****     S Y N S E T
1040 REM              -----------
1060 REM
1080 REM
1100 REM
1120 FILEVN#1:'RXA1:SYCOR.TS'
1140 FOR K=1 TO 20
1160 PRINT#1:0
1180 NEXT K
1200 CLOSE#1
1220 PRINT 'YOU CAN NOW RUN ''SYNONY'' BY TYPING:'
1240 PRINT '         OLD RXA1:SYNONY'
1260 PRINT 'AND THEN:'
1280 PRINT '         RUN'
1300 END
```

WTDAVG

This program computes the weighted average of a group of grades. The user must tell the program the number of grades that will be included in each average and the relative weight that each is to have in the final computation. Further instructions are contained within the program itself, and use of this program is discussed in Module 5-A.

```
LIST

WTDAVG  BA    3.0     30-DEC-75

1000 REM              -----------
1010 REM     ****     W T D A V G
1020 REM              -----------
1030 REM
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM     ****     VARIABLE DIRECTORY
1090 REM
1100 REM     VARIABLE        USAGE
1110 REM     --------        -----
1120 REM       A$      GENERAL ALPHAMERIC USER INPUT
1130 REM       D       DIVISOR FOR WEIGHTED AVERAGE
1140 REM       G(K)    GRADES
1150 REM       I       STUDENT NUMBER INDEX
1160 REM       K       GENERAL FOR-NEXT LOOP INDEX
1170 REM       K$      RECAPITULATION TYPE
1180 REM       N       NUMBER OF GRADES
1190 REM       T       FINAL GRADE OR TOTAL OF WEIGHTS
1200 REM       W(K)    WEIGHTS OF GRADES
1210 REM
1220 REM     ****     DECLARATIONS
1230 REM
1240 DIM G(100),W(100)
1250 LET I=1
1260 REM              -----------------------
1270 REM     ****     M A I N   P R O G R A M
1280 REM              -----------------------
1290 REM
1300 REM
1310 REM     ****     INSTRUCTIONS QUERY
1320 REM
1330 GOSUB 2310
1340 PRINT 'WEIGHTED AVERAGING'
1350 PRINT '------------------'
1360 PRINT
1370 PRINT
1380 PRINT 'DO YOU WISH TO SEE THE INSTRUCTIONS (''YES'' OR ''NO'')';
1390 INPUT A$
1400 PRINT
1410 IF POS(A$,'Y',1)<>0 THEN 1470
1420 IF POS(A$,'N',1)<>0 THEN 1480
1430 IF A$='QUIT' THEN 2640
1440 PRINT '     PLEASE INPUT ''YES'', ''NO'', OR ''QUIT''.  YOUR ';
1450 PRINT 'CHOICE';
1460 GOTO 1390
1470 GOSUB 2360
1480 REM
1490 REM     ****     NUMBER OF GRADES INPUT
1500 PRINT
1510 PRINT 'HOW MANY GRADES DO YOU HAVE FOR EACH STUDENT';
1520 GOSUB 2670
1530 IF A<>-99999     THEN 1590
1540 IF A$='GRADES'   THEN 1500
1550 IF A$<>'WEIGHTS' THEN 1670
1560 IF N>1 THEN 1720
1570 PRINT '     WE FIRST NEED TO KNOW HOW MANY GRADES YOU WILL ENTER.'
1580 GOTO 1500
1590 IF A=-88888 THEN 1500
1600 IF A=-77777 THEN 1500
1610 IF A<>INT(A) THEN 1670
1620 IF A<0       THEN 1670
1630 IF A>=2      THEN 1700
1640 PRINT
1650 PRINT '     YOU NEED AT LEAST TWO NUMBERS TO FIND AN AVERAGE.'
1660 GOTO 1500
1670 PRINT
1680 PRINT '     PLEASE INPUT A POSITIVE INTEGER.'
1690 GOTO 1500
1700 LET N=A
1710 REM     ****     WEIGHT INPUTS
1720 PRINT
1730 PRINT 'INPUT YOUR RELATIVE WEIGHTS FOR EACH GRADE BELOW:'
1740 FOR K=1 TO N
1750 PRINT '     WEIGHT FOR GRADE #'; K;
1760 GOSUB 2670
1770 IF A<>-99999    THEN 1820
1780 IF A$='GRADES'  THEN 1500
1790 IF A$='WEIGHTS' THEN 1720
1800 PRINT '     POSITIVE WEIGHTS ONLY, PLEASE!'
1810 GOTO 1750
1820 IF A<>-88888 THEN 1880
1830 LET K$='WEIGHT'
1840 IF K=1 THEN 1750
1850 PRINT 'THE WEIGHTS YOU HAVE ENTERED SO FAR ARE:'
1860 GOSUB 3070
1870 GOTO 1750
1880 IF A=-77777 THEN 1720
1890 IF A<0 THEN 1800
1900 LET W(K)=A
1910 NEXT K
1920 REM     ****     GRADE INPUTS
1930 PRINT
1940 PRINT 'INPUT YOUR GRADES FOR STUDENT #'; I; 'BELOW:'
1950 FOR K=1 TO N
1960 PRINT '     GRADE #'; K;
1970 GOSUB 2670
1980 IF A<>-99999    THEN 2010
1990 IF A$='GRADES'  THEN 1500
2000 IF A$='WEIGHTS' THEN 1720
2010 IF A<>-88888    THEN 2070
2020 LET K$='GRADES'
2030 IF K=1 THEN 1960
```

continued on next page

```
2040 PRINT "THE GRADES YOU HAVE ENTERED FOR STUDENT #"; I; "SO FAR ARE:"
2050 GOSUB 3070
2060 GOTO 1960
2070 IF A=-77777 THEN 1930
2080 LET G(K)=A
2090 NEXT K
2100 REM    ****    WEIGHTED AVERAGE CALCULATION
2110 REM
2120 LET T=0
2130 LET D=0
2140 FOR K=1 TO N
2150 LET T=T+G(K)*W(K)
2160 LET D=D+W(K)
2170 NEXT K
2180 PRINT
2190 PRINT "THE WEIGHTED AVERAGE OF STUDENT #"; I; "'S GRADES  = "; T/D
2200 PRINT
2210 LET I=I+1
2220 GOTO 1930
2230 REM
2240 REM            ---------------------
2250 REM    ****    S U B R O U T I N E S
2260 REM            ---------------------
2270 REM
2280 REM
2290 REM    ****    SCREEN CLEARER
2300 REM
2310 PRINT
2320 RETURN
2330 REM
2340 REM    ****    INSTRUCTIONS
2350 REM
2360 GOSUB 2310
2370 PRINT "THIS PROGRAM COMPUTES WEIGHTED AVERAGES OF SETS OF ";
2380 PRINT "GRADES.  FIRST IT"
2390 PRINT "WILL ASK YOU HOW MANY GRADES YOU WILL ENTER PER STUDENT. ";
2400 PRINT " THEN IT WILL"
2410 PRINT "ASK THE RELATIVE WEIGHTS TO ASSIGN EACH GRADE.  FOR A ";
2420 PRINT "REGULAR AVERAGE,"
2430 PRINT "ALL THE RELATIVE WEIGHTS ARE ""1"".  IF YOU WANT TO ";
2440 PRINT "WEIGHT A GRADE TWICE"
2450 PRINT "AS HEAVILY AS NORMAL, ENTER ""2"", ETC.  YOU WILL THEN ";
2460 PRINT "ENTER ALL YOUR"
2470 PRINT "GRADES FOR ONE STUDENT, THEIR WEIGHTED AVERAGE WILL BE ";
2480 PRINT "DISPLAYED, AND"
2490 PRINT "THE PROGRAM WILL RECYCLE FOR ANOTHER STUDENT'S GRADES.  ";
2500 PRINT "YOU MAY ENTER A"
2510 PRINT "VALID OPTION TO ANY INPUT QUERY.  THESE OPTIONS ARE ";
2520 PRINT """GRADES"", ""HELP"",";
2530 PRINT """INSTR"", ""QUIT"", ""RESTART"", AND ""WEIGHTS"".  ";
2540 PRINT """HELP"" PRINTS AN EXPLANATION"
2550 PRINT "OF EACH."
2560 PRINT
2570 PRINT "(TYPE ""Y"" AND PRESS ""RETURN"" WHEN YOU HAVE FINISHED ";
2580 PRINT "READING.)  READY";
2590 INPUT A$
2600 IF POS(A$,"Y",1)<>0 THEN 2620
2610 GOSUB 2680
2620 GOSUB 2310
2630 RETURN
2640 REM
2650 REM    ****    RESPONSE DECODER
2660 REM
2670 INPUT A$
2680 IF A$="GRADES"  THEN 2930
2690 IF A$="HELP"    THEN 2970
2700 IF A$="INSTR"   THEN 3020
2710 IF A$="QUIT"    THEN 3370
2720 IF A$="RESTART" THEN 2890
2730 IF A$="WEIGHTS" THEN 2930
2740 FOR K1=1 TO LEN(A$)
2750 IF ASC(SEG$(A$,K1,K1))=46 THEN 2780
2760 IF ASC(SEG$(A$,K1,K1))<48 THEN 2820
2770 IF ASC(SEG$(A$,K1,K1))>57 THEN 2820
2780 NEXT K1
2790 LET A=VAL(A$)
2800 RETURN
2810 REM    ****    INVALID OPTION
2820 PRINT
2830 PRINT "     INVALID ENTRY -- TRY AGAIN OR ENTER ""HELP"".  YOUR";
2840 PRINT "YOUR CHOICE";
2850 GOTO 2670
2860 REM
2870 REM    ****    "RESTART" ENTERED
2880 REM
2890 LET A=-77777
2900 RETURN
2910 REM    ****    "GRADES" OR "WEIGHTS" ENTERED
2920 REM
2930 LET A=-99999
2940 RETURN
2950 REM    ****    "HELP" ENTERED
2960 REM
2970 GOSUB 3240
2980 LET A=-88888
2990 RETURN
3000 REM    ****    "INSTR" ENTERED
3010 REM
3020 GOSUB 2360
3030 GOTO 2980
3040 REM
3050 REM    ****    RECAPITULATION PRINTER
3060 REM
3070 IF K=1 THEN 3200
3080 FOR K1=1 TO K STEP 5
3090 FOR K2=K1 TO K1+4
3100 IF K2>=K THEN 3170
3110 PRINT "     ("; SEG$("  ",1,2-LEN(STR$(K2))); STR$(K2); ")  ";
3120 IF K$="GRADES" THEN 3150
3130 PRINT W(K2);
3140 GOTO 3160
3150 PRINT G(K2);
3160 NEXT K2
3170 PRINT
3180 PRINT
```

continued on next column

```
3190 NEXT K1
3200 RETURN
3210 REM
3220 REM    ****    HELP MESSAGE
3230 REM
3240 GOSUB 2310
3250 PRINT "THE VALID OPTIONS ARE AS FOLLOWS:"
3260 PRINT "     GRADES      CHANGE THE NUMBER OF GRADES PER STUDENT"
3270 PRINT "     HELP        DISPLAY THIS MESSAGE"
3280 PRINT "     INSTR       DISPLAY THE INSTRUCTIONS"
3290 PRINT "     QUIT        TERMINATE THE PROGRAM"
3300 PRINT "     RESTART     ERASE THE CURRENT SET OF WEIGHTS OR ";
3310 PRINT "GRADES AND"
3320 PRINT "                 RESTART THE ENTERING PROCEDURE "
3330 PRINT "     WEIGHTS     CHANGE THE WEIGHTS ASSIGNED FOR EACH ";
3340 PRINT "GRADE"
3350 PRINT "THE WEIGHT FOR EACH GRADE MUST BE GREATER THAN 0."
3360 GOTO 2560
3370 END
```

Appendix B DECUS Program Submission Forms

DECUS LIBRARY
PROGRAM SUBMISSION INFORMATION

Programs should be submitted to:

DECUS Program Librarian Digital Equipment Computer Users Society 146 Main Street Maynard, Massachusetts U.S.A. 01754	or	DECUS Executive Secretary Case Postale 340 1211 Geneva 26/ Switzerland

The following material MUST be included:

(1) Completed submittal form

Read the following notes which explain the form.

Section A

(1) Object Computer(s) - computer(s) on which the program runs.
Source Computer - computer on which program was assembled (if different).
(2) File Name - mnemonic or acronym of 6 characters (8 for PDP-12) for mass storage purposes.
Version No. - indicate version or development level. If unspecified DECUS will assume version No. 1.
(3-7) Self-explanatory.
(8) Category Codes - indicate the category or categories best suited for the category index of the library catalog.
(9) Monitor - if the program runs under a monitor, all relevant details must be specified.
(10-14) Self-explanatory
(15) Please indicate if major revision or development is planned, with estimate of completion date.

Section B

The submission of an assembly (Pass 3) listing is optional but desirable; short listings may be incorporated into the write-up. Other acceptable material includes flow-charts, cross referenced listings, core maps or any other relevant documentation. The abstract must be written in English but full documentation may be in any language.

Section C

The authorization at the bottom of the submission form must be signed by the person having legal right and interest in the submitted program.

(2) Abstract

An abstract (in English) of up to 100 words must be attached. This will be used in the preparation of the DECUS Library Catalog entry.

(3) Write-up

It is requested that documents be suitable for direct reproduction. Clear operating and loading instructions must be part of any document submitted. Where applicable a printed copy of the tape file index, including a brief description of each file function, would be helpful.

(4) Paper Tape

All material should be fully labelled with program name, version, starting address (where applicable) and tape format (ASCII, binary, etc.). Source tapes should be submitted whenever possible.

and/or

(5) DECtape/LINCtape/Magtape

Attach to each tape a printed index of tape file contents. Specify mark track format used. Source files should be submitted whenever possible.

PROGRAM REVISIONS

Revisions to existing DECUS or DEC programs should be accompanied by a program revision submission form (attached)

EN-1146B-07-R275-(369) January 1975

DECUS LIBRARY
PROGRAM REVISION SUBMISSION

Form to be used for modifications or revisions to existing DEC or DECUS software.

A. GENERAL INFORMATION

1. Object Computer(s) ____________ Source Computer (if different) ____________
2. Original File Name and Title ____________
   ____________ DECUS or DEC No. ____________
3. Original Author ____________
4. Revising Author ____________
5. Affiliation ____________
6. Address ____________
   ____________ Country ____________

B. CHANGE INFORMATION

Please specify any changes to the following:

1. Category ____________
2. Monitor/Operating System* ____________ DEC No.* ____________
3. Core Storage Required ____________ Starting Address* ____________
4. Hardward Required ____________
5. Other Software Required ____________ DEC or DECUS No.* ____________
6. Restrictions, Deficiencies, Problems ____________

C. REASON(s) FOR REVISION

1. Debug, correct known problem ☐
2. Extend to handle new or different configurations ☐
3. Operate under different monitor or new system ☐
4. Increased operational efficiency ☐
5. Operate on different processor ☐ Specify ____________
6. Other (please specify) ____________

D. MATERIAL SUBMITTED

Documentation

All revisions should include a detailed statement of the changes made to the existing program.

Revised Abstract ☐ Revised Write-up ☐ New Listing ☐

Paper Tape

Object Binary ☐ Binary Patch ☐ Object ASCII ☐ Source ☐ Other ____________

DECtape ☐ LINCtape ☐ Mark Track Format ________ Magtape: 7 Track ☐ 9 Track ☐ BPI ________

Specify Format/System (e.g. OS/8, LAP4, DIAL, DOS-11, DOS-15, etc.) ____________

Object Files ☐ Source Files ☐ Documentation Files ☐ Other ____________

E. AUTHORIZATION

I, the undersigned, give full permission to DECUS to publish information regarding this revision and to reproduce and distribute this revision in full or in part to all interested parties, in accordance with the then standard policies of DECUS for reproduction and distribution of programs submitted to DECUS. I further warrant and represent that I have good and sufficient title and all rights and interest in and to the revision to grant such permission to DECUS.

Date ____________ Signed ____________

*Where Applicable

Appendix C
Answers To Exercises

2.

```
RUN

FIRST    BA     3.0       04-FEB-76

 15
 9
 36
 4

READY
```

5.

```
10 PRINT 11; 22; 33; 44
20 PRINT 1/3; 2/3; 1; 4/3
30 PRINT 1/6; 5/6; 1/8; 7/8
40 PRINT -1, -2, -3, -4
99 END

READY
```

6.

```
10 PRINT ,,.5
20 PRINT ,.25,,.75
30 PRINT 0,,,,1
40 PRINT ,-.25,,-.75
50 PRINT ,,-.5
99 END

READY

10 PRINT ,,.5,,,,.25,,.75,,0,,,,1,,-.25,,-.75,,,,-.5
99 END
```

7.

```
RUNNH
COMPUTERS DO ARITHMETIC LIKE THIS:
3+4-5 = 2      3+4*5 = 23     3+4/5 = 3.8
3-4+5 = 4      3-4*5 =-17     3-4/5 = 2.2
3*4+5 = 17     3*4-5 = 7      3*4/5 = 2.4
3/4+5 = 5.75   3/4-5 =-4.25   3/4*5 = 3.75
6+5-4*3/2 = 5

READY
```

8.

12∧(4/2)	=	144
5^5	=	3125
3/4∧2	=	0.1875
(3/4)∧2	=	0.5625
3/(4∧2)	=	0.1875
10^{10-6}	=	10000
(2+6)∧(4-2)	=	64
7∧1	=	7
7∧0	=	1
0∧8	=	0

10.

```
READY
10 LET X=3
20 LET X=5
30 LET X=7
40 PRINT X
99 END
RUNNH
 7

READY
```

```
10 LET X=3
20 LET Y=5
30 LET Z=7
40 PRINT X;Y;Z
99 END

READY
RUNNH
 3  5  7

READY
```

11.

```
10 LET A=3
20 LET B=4
30 PRINT A-B
99 END

READY
RUNNH
-1

READY
```

```
10 LET A=3
20 LET B=4
30 PRINT A/B
99 END
RUNNH
 0.75

READY
```

12.

```
5 PRINT "BASE AND HEIGHT";
10 INPUT B,H
20 PRINT "BASE","HEIGHT","AREA"
30 PRINT B,H,0.5*B*H
99 END

READY
RUNNH
BASE AND HEIGHT?7.31,6.04
```

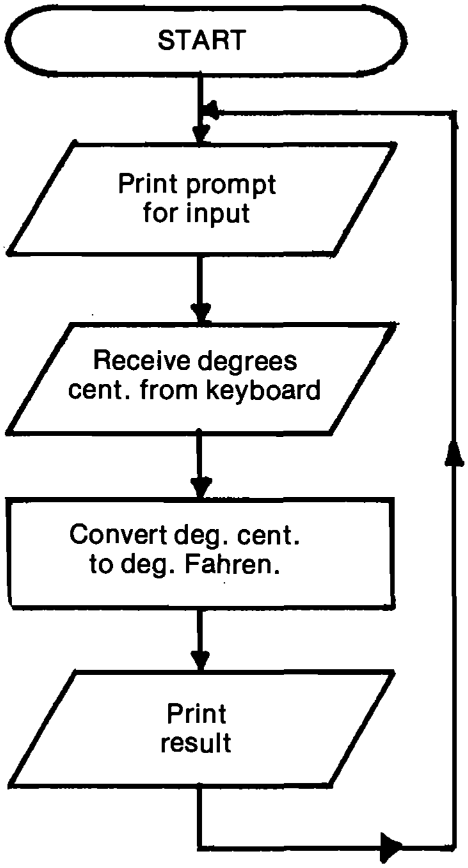
continued on next column

```
BASE            HEIGHT          AREA
 7.31            6.04            22.0762

READY
RUNNH
BASE AND HEIGHT?82,127
BASE            HEIGHT          AREA
 82              127             5207

READY
RUNNH
BASE AND HEIGHT?5E4,9E5
BASE            HEIGHT          AREA
 50000           900000          .225000E+011

READY
RUNNH
BASE AND HEIGHT?23.491,17.260
BASE            HEIGHT          AREA
 23.491          17.26           202.727

READY
```

13.

```
10 PRINT "CENTIGRADE TEMPERATURE";
20 INPUT C
25 REM
30 LET F =  (9/5) * C + 32
35 REM
40 PRINT C; "DEG. CENT. = ";F;"DEG. FAHREN."
50 PRINT
55 REM
60 GOTO  10
70 REM
99 END

READY
RUNNH
CENTIGRADE TEMPERATURE?100
 100 DEG. CENT. =  212 DEG. FAHREN.

CENTIGRADE TEMPERATURE?37
 37 DEG. CENT. =  98.6 DEG. FAHREN.

CENTIGRADE TEMPERATURE?6.8
 6.8 DEG. CENT. =  44.24 DEG. FAHREN.

CENTIGRADE TEMPERATURE?0
 0 DEG. CENT. =  32 DEG. FAHREN.

CENTIGRADE TEMPERATURE?-40
-40 DEG. CENT. = -40 DEG. FAHREN.

CENTIGRADE TEMPERATURE?-100
-100 DEG. CENT. = -148 DEG. FAHREN.

CENTIGRADE TEMPERATURE?-273.15
-273.15 DEG. CENT. = -459.67 DEG. FAHREN.

CENTIGRADE TEMPERATURE?^C
READY
```

14.



15.

```
RUNNH
THIS PROGRAM WILL FIND THE AREA OF A
CIRCLE FOR WHICH THE RADIUS IS ENTERED.

ENTER BELOW THE RADIUS OF A CIRCLE:

YOUR FIRST CIRCLE'S RADIUS?3

RADIUS         AREA
 3              28.26

YOUR NEXT CIRCLE'S RADIUS?4

RADIUS         AREA
 4              50.24

YOUR NEXT CIRCLE'S RADIUS?^C
```

17. The maximum number of lines that the CLASSIC screen can display at once is 12.

23.

Before		Statement	After	
K	25	30 LET K=K+L	K	26
E	6	40 LET E=E+2	E	8
N	4.2	200 LET N=N*5	N	21
X	-10	235 LET X=X+5	X	-5
P	0	280 LET P=P-20	P	-20
Q	-3.1	310 LET Q=15+Q	Q	11.9
L	5	325 LET L=L+L+L	L	15
B	7	340 LET B= -B+B	B	0

24.

Statement	A	B	C	Remarks
10 LET A=1	1			These statements are done once.
17 LET B=1	1	1		
25 LET C=A+B	1	1	2	First time through loop.
30 PRINT A	1	1	2	
36 LET A=B	1	1	2	
43 LET B=C	1	2	2	
50 GO TO 25				
25 LET C=A+B	1	2	3	Second time through loop.
30 PRINT A	1	2	3	
36 LET A=B	2	2	3	
43 LET B=C	2	3	3	
50 GO TO 25				
25 LET C=A+B	2	3	5	Third time through loop.
30 PRINT A	2	3	5	
36 LET A=B	3	3	5	
43 LET B=C	3	5	5	
50 GO TO 25				
25 LET C=A+B	3	5	8	Fourth time through loop.
30 PRINT A	3	5	8	
36 LET A=B	5	5	8	
43 LET B=C	5	8	8	
50 GO TO 25				

25.

```
READY
10 LET X=1
20 PRINT X
30 LET X=X+2
40 GOTO 20
99 END
RUNNH
 1
 3
 5
```

```
10 LET E=2
20 PRINT E
30 LET E=E+2
40 GOTO 20
99 END
RUNNH
 2
 4
 6
 8
```

continued on next page

```
7
9
11
13
15
17
19
21^C
READY
```

```
10
12
14
16
18
20
22 ^C
READY
```

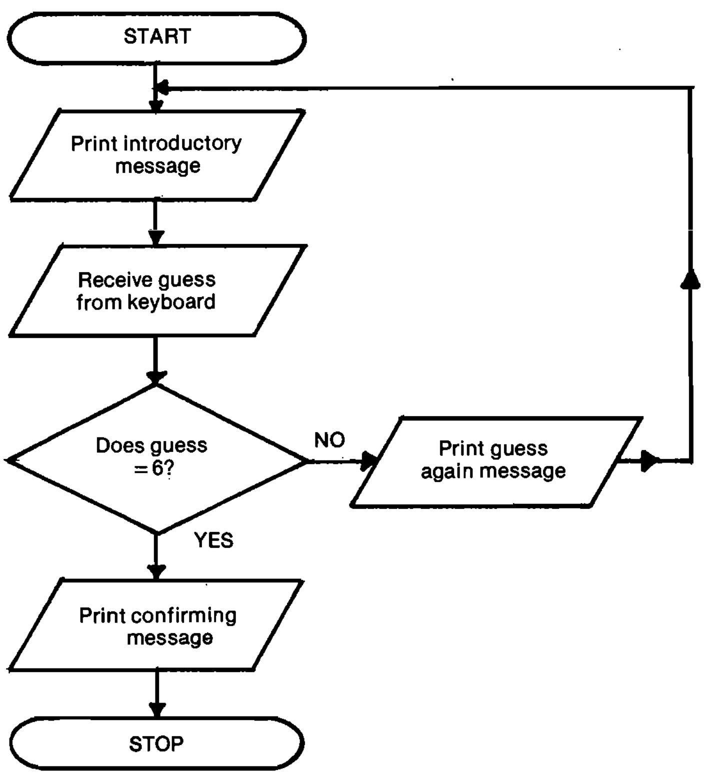
26.

```
10 LET J=0
20 PRINT J
30 LET J=J+1
40 GOTO 20
99 END
```

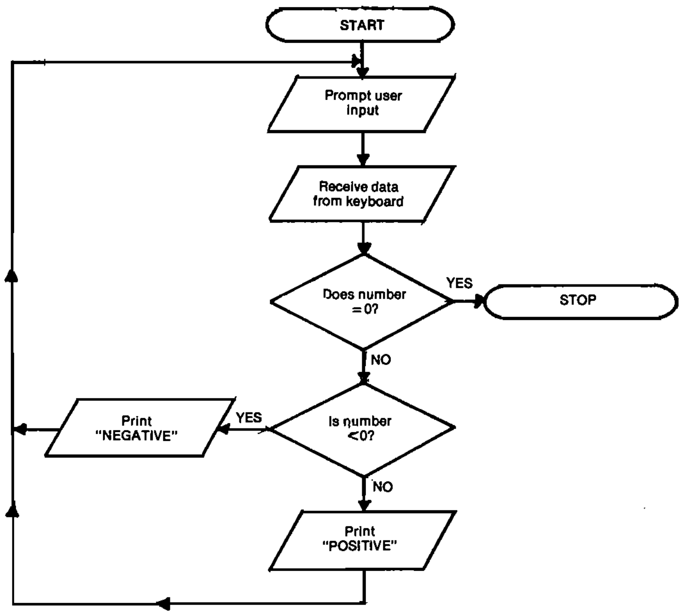
```
10 LET P=1
20 PRINT P
30 LET P=P*2
40 GOTO 20
99 END
```

```
10 LET S=36
20 LET S=S/3
30 PRINT S
40 GOTO 20
99 END
```

27.



28.



29.

```
10 PRINT "NUMBERS TO BE TESTED";
20 INPUT A,B
30 LET T=B
40 LET T=T-A
50 IF T=0 THEN 90
60 IF T>0 THEN 40
70 PRINT A; "IS NOT A FACTOR OF";B
80 GOTO 10
90 PRINT A;"IS A FACTOR OF";B
95 GOTO 10
99 END
```

30.

First Number	Second Number	Is the First a Factor of the Second?
8	64	Yes
6	44	No
12	576	Yes
42	840	Yes
103	103	Yes
13	1276	No
11	6336	Yes
231	591	No
208	5200	Yes
184	1417	No
276	826	No
55	1870	Yes

31.

```
10 PRINT "NUMBERS TO BE TESTED";
20 INPUT A,B
30 LET T=B
40 LET T=T-A
50 IF T=0 THEN 90
60 IF T>0 THEN 40
70 PRINT A; "IS NOT A FACTOR OF";B
80 GOTO 91
90 PRINT A;"IS A FACTOR OF";B
91 PRINT
93 PRINT "DO YOU HAVE MORE NUMBERS";
95 INPUT A$
97 IF A$="YES" THEN 10
99 END

READY
RUNNH
NUMBERS TO BE TESTED?5,30
 5 IS A FACTOR OF 30

DO YOU HAVE MORE NUMBERS?YES
NUMBERS TO BE TESTED?5,31
 5 IS NOT A FACTOR OF 31

DO YOU HAVE MORE NUMBERS?NO

READY
```

32. Statements 25 and 30 make up the body of the loop in this program.

33.

```
READY
10 PRINT "RADIUS","VOLUME"
20 FOR R=1 TO 10
30 PRINT R,(4/3)*3.14*R^3
40 NEXT R
99 END
RUNNH
RADIUS        VOLUME
 1             4.18666
 2             33.4933
 3             113.04
 4             267.946
 5             523.333
 6             904.32
 7             1436.02
 8             2143.57
 9             3052.08
 10            4186.66

READY
```

34.

FOR Statement	Variable	Set of Values for the Variable
FOR N = 1 TO 6	N	[1,2,3,4,5,6]
FOR C = 0 TO 5	C	[0,1,2,3,4,5]
FOR W = -3 TO 0	W	[-3,-2,-1,0]
FOR E = 12 TO 12	E	[12]
FOR T = 7 TO 5	T	Empty
FOR X = .5 TO 2.5	X	[.5,1.5,2.5]
FOR Y = 1 TO 2.5	Y	[1,2]
FOR Z = .5 TO 3	Z	[.5,1.5,2.5]

35.

FOR Statement	Values of the variable
FOR T = 0 TO 6 STEP 3	T = [0,3,6]
FOR N = 1 TO 5 STEP 1	N = [1,2,3,4,5]
FOR K = 100 TO 130 STEP 10	K = [100,110,120,130]
FOR X = 0 TO 1 STEP .25	X = [0,.25,.5,.75,1]
FOR E = 0 TO 0 STEP 2	E = [0]
FOR B = 3 TO 0 STEP -1	B = [3,2,1,0]

36.

```
10 PRINT "RADIUS","SURFACE AREA"
20 FOR R=10 TO 100 STEP 10
30 PRINT R,4*3.14*R^2
40 NEXT R
99 END

READY
RUNNH
RADIUS        SURFACE AREA
 10            1256
 20            5024
 30            11304
 40            20096
 50            31400
 60            45216
 70            61544
 80            80384
 90            101736
 100           125600

READY
```

37.

Initial value	Terminal value	Step value	Index values
0	1	0.2	[0,.2,.4,.6,.8,1]
10	0	3	Empty
2	5	2	[2,4]
6	6	3	[6]
0.0010	0.0013	0.0001	[.001,.0011,.0012,.0013]
8	8	-1	[8]
-3	-4	-0.3	[-3,-3.3,-3.6,-3.9]
-4	-3	-0.3	Empty
926	1852	463	[926,1389,1852]
0.01	-0.01	-0.005	[.01,.005,0,-.005,-.01]

38.

```
100 FOR Y=-5 TO 5
110 PRINT
120 LET X=Y^2
130 FOR C=-35 TO 36
140 IF C>=X THEN 170
150 PRINT " ";
160 NEXT C
170 PRINT "*";
180 NEXT Y
999 END
RUNNH

                                  *
                         *
                  *
             *
          *
         *
          *
             *
                  *
                         *
                                  *
READY
```

39.

```
100 PRINT "NUMBER"
110 PRINT "OF SCORES","AVERAGE"
120 PRINT
130 READ N
140 LET S=0
150 FOR K=1 TO N
160 READ T
170 LET S=S+T
180 NEXT K
190 PRINT N,S/N
200 GOTO 130
900 DATA 3,82,88,97
```

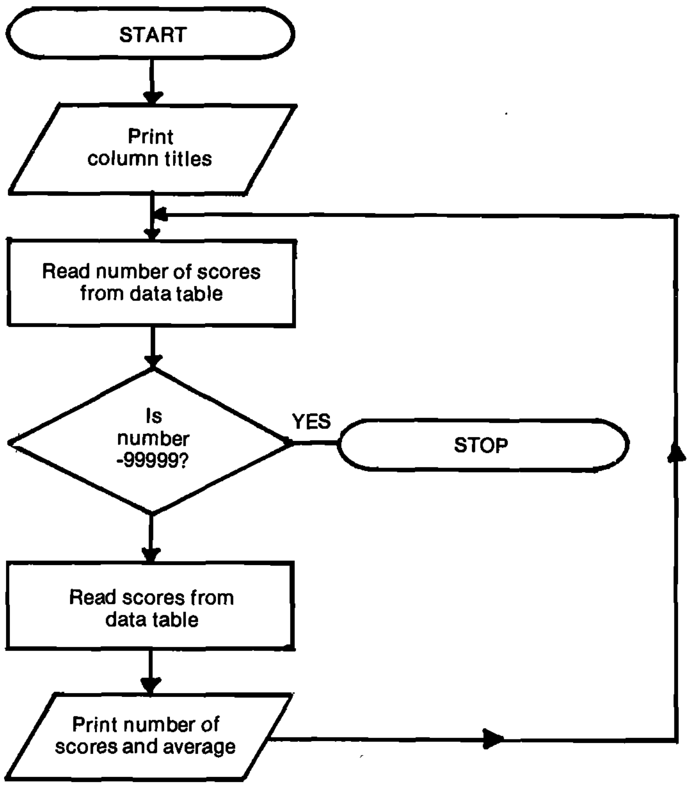
continued on next column

```
910 DATA 5,66,78,71,82,75
920 DATA 4,82,86,100,91
930 DATA 4,72,82,73,82
940 DATA 6,61,73,67,80,84,79
999 END

READY
RUNNH
NUMBER
OF SCORES      AVERAGE

 3              89
 5              74.4
 4              89.75
 4              77.25
 6              74

DA AT LINE 00130

READY
```

40.

```
100 PRINT "NUMBER"
110 PRINT "OF SCORES","AVERAGE"
120 PRINT
130 READ N
135 IF N=-99999 THEN 999
140 LET S=0
150 FOR K=1 TO N
160 READ T
170 LET S=S+T
180 NEXT K
190 PRINT N,S/N
200 GOTO 130
900 DATA 3,82,88,97
910 DATA 5,66,78,71,82,75
920 DATA 4,82,86,100,91
930 DATA 4,72,82,73,82
940 DATA 6,61,73,67,80,84,79
950 DATA -99999
999 END

READY
RUNNH
NUMBER
OF SCORES      AVERAGE

 3              89
 5              74.4
 4              89.75
 4              77.25
 6              74

READY
```

39.



41.

```
100 PRINT "CODED MESSAGE";
110 INPUT C
120 IF C>0 THEN 130
123 PRINT
125 PRINT
127 GOTO 100
130 IF C<=26 THEN 160
140 PRINT " ";
150 GOTO 110
160 RESTORE
170 FOR K=1 TO C
180 READ L$
190 NEXT K
200 PRINT L$;
210 GOTO 110
500 DATA "A","B","C","D","E","F","G"
510 DATA "H","I","J","K","L","M","N"
520 DATA "O","P","Q","R","S","T","U"
530 DATA "V","W","X","Y","Z"
999 END

READY
RUNNH
CODED MESSAGE?20,8,9,19,32,9,19,28,20,8,5,34,13,5,19,19,1,7,5,0
THIS IS THE MESSAGE

CODED MESSAGE?^C
READY
```

42.

Incorrect statement	Reason
10 READ,A,B,C	Extra comma
20 READ,XY	Extra comma and invalid variable name
30 REED P,Q,R,S,T	READ spelled incorrectly
40 READ A+B	Calculations not allowed in READ statements
50 READ I;J;K	Commas required between variables
60 READ AA,BB	Invalid variable names
70 READ ABC	Commas omitted
80 READ 3.14	Constants not allowed in READ statements
120 DATA 1/2,2/3,3/4	Calculations not allowed in DATA statements
130 DATA A,B,C,D,E	Variables not allowed in DATA statements
140 DATA,3.7,2.9	Extra comma

43.

Subscripted variable	Value	Subscripted variable	Value
A(1)	8	A(2*I)	-6
A(I)	8	A(I+J)	10
A(K)	10	A(I+2)	10
A(X)	13	A(2*J-1)	10
B(I)	3.7	A(X-3)	8
B(3)	3	A(X-K+J)	10
B(J)	9.2	A(J*K-X)	-6
C(J)	4	A(C(2))	13
B(1+I)	9.2	A(B(C(2)-1))	10

44.

```
10 FOR N=1 TO 4
20 LET P(N)=2 N
30 NEXT N
```

P(1)	2	P(3)	8
P(2)	4	P(4)	16

```
70 LET F(1)=1
75 FOR K= TO 6
80 LET F(K) =K*F(K-1)
85 NEXT K
```

F(1)	1	F(4)	24
F(2)	2	F(5)	120
F(3)	6	F(6)	720

45.

```
100 PRINT "ORIGINAL DATA:"
110 FOR K=1 TO 10
120 READ N(K)
130 PRINT N(K);
140 NEXT K
150 PRINT
160 PRINT
170 FOR K=1 TO 5
180 LET T=N(K)
190 LET N(K)=N(11-K)
200 LET N(11-K)=T
210 NEXT K
220 PRINT "INVERTED DATA:"
230 FOR K=1 TO 10
240 PRINT N(K);
250 NEXT K
500 DATA 23,4,35,32,19,7,26,8,14,13
999 END

READY
RUNNH
ORIGINAL DATA:
 23   4   35   32   19   7   26   8   14   13

INVERTED DATA:
 13   14   8   26   7   19   32   35   4   23
READY
```

46.

```
80 DIM N(100)
90 READ S
100 PRINT "UNSORTED DATA:"
110 FOR K=1 TO S
120 READ N(K)
130 PRINT N(K); \ IF K<>12 THEN 140 \PRINT
140 NEXT K
150 PRINT
160 PRINT
170 FOR K1=1 TO S-1
180 FOR K2=K1+1 TO S
190 IF N(K1)<=N(K2) THEN 230
200 LET T=N(K1)
210 LET N(K1)=N(K2)
220 LET N(K2)=T
230 NEXT K2
240 NEXT K1
250 PRINT "SORTED DATA:"
260 FOR K=1 TO S
270 PRINT N(K); \ IF K<>12 THEN 280 \ PRINT
280 NEXT K
500 DATA 25
510 DATA 21,60,44,20,72,67,12,63,76
520 DATA 27,92,54,39,64,52,22,65,46,55
530 DATA 34,68,23,96,43
540 DATA 64
999 END
```

continued on next column

```
READY
RUNNH
UNSORTED DATA:
 21  60  44  20  72  67  12  63  76  27  92  54
 39  64  52  22  65  46  55  34  68  23  96  43  64

SORTED DATA:
 12  20  21  22  23  27  34  39  43  44  46  52
 54  55  60  63  64  64  65  67  68  72  76  92  96
READY
```

47.

```
LISTNH
100 DIM C(8),T(8,2)
110 FOR K1=1 TO 8
120 READ C(K1)
130 FOR K2=0 TO 2
140 LET T(K1,K2)=0
150 NEXT K2
160 NEXT K1
170 READ N
180 FOR K1=1 TO N
190 FOR K2=1 TO 8
200 READ R
210 IF R>0 THEN 240
220 LET T(K2,0)=T(K2,0)+1
230 GOTO 280
240 IF R=C(K2) THEN 270
250 LET T(K2,1)=T(K2,1)+1
260 GOTO 280
270 LET T(K2,2)=T(K2,2)+1
280 NEXT K2
290 NEXT K1
300 PRINT "QUESTION","CORRECT","INCORRECT"
310 PRINT "NUMBER","RESPONSES","RESPONSES","OMITTED"
320 FOR K1=1 TO 8
330 PRINT K1,
340 FOR K2=2 TO 0 STEP -1
350 PRINT T(K1,K2),
360 NEXT K2
370 PRINT
380 NEXT K1
500 DATA 2,3,3,2,1,2,1,3
510 DATA 10
520 DATA 2,1,3,0,3,3,2,2
530 DATA 2,3,3,2,1,1,2,2
540 DATA 1,3,2,3,1,2,0,0
550 DATA 2,1,2,1,3,2,3,3
560 DATA 2,2,3,0,1,3,3,2
570 DATA 2,3,2,1,3,2,0,2
580 DATA 1,3,3,2,1,3,1,0
590 DATA 2,1,2,0,0,1,3,3
600 DATA 2,2,3,1,1,2,3,2
610 DATA 2,3,2,3,3,2,3,1
999 END

READY
RUNNH
QUESTION        CORRECT         INCORRECT
NUMBER          RESPONSES       RESPONSES       OMITTED

 1               8               2               0
 2               5               5               0
 3               5               5               0
 4               2               5               3
 5               5               4               1
 6               5               5               0
 7               1               7               2
 8               2               6               2

READY
```

48.

```
LISTNH
10 REM  ***  SCORE
20 REM
30 REM     THIS PROGRAM SCORES AN EIGHT QUESTION MULTIPLE CHOICE
40 REM  TEST AND TALLIES THE NUMBER OF STUDENTS WHO ANSWERED EACH
50 REM  QUESTION CORRECTLY (STORED IN T(Q,2)), INCORRECTLY (STORED
60 REM  IN T(Q,1)), AND NOT AT ALL (T(Q,0)).
70 REM
100 DIM C(8),T(8,2)
102 REM     THE CORRECT ANSWERS FOR THE TEST ARE STORED IN LIST 'C'.
104 REM  THE RESPONSE TALLIES ARE STORED IN ARRAY 'T'.
105 REM
106 REM  ***  INITIALIZING ROUTINE
108 REM
110 FOR K1=1 TO 8
120 READ C(K1)
130 FOR K2=0 TO 2
140 LET T(K1,K2)=0
150 NEXT K2
160 NEXT K1
165 REM
170 READ N
172 REM  'N' IS THE NUMBER OF TESTS TO BE SCORED.
174 REM
176 REM  *** TALLYING ROUTINE
178 REM
180 FOR K1=1 TO N
190 FOR K2=1 TO 8
200 READ R
205 REM  'R' IS A RESPONSE TO QUESTION NUMBER 'K2'.
210 IF R>0 THEN 240
213 REM
215 REM     THE NEXT TWO STATEMENTS ARE EXECUTED ONLY IF NO
217 REM  RESPONSE WAS GIVEN.
220 LET T(K2,0)=T(K2,0)+1
225 REM
230 GOTO 280
235 REM
240 IF R=C(K2) THEN 270
243 REM
245 REM     THE NEXT TWO STATEMENTS ARE EXECUTED ONLY IF AN
247 REM  INCORRECT RESPONSE WAS GIVEN.
250 LET T(K2,1)=T(K2,1)+1
255 REM
260 GOTO 280
263 REM
265 REM     THE NEXT STATEMENT IS EXECUTED ONLY IF A CORRECT
267 REM  RESPONSE WAS GIVEN.
270 LET T(K2,2)=T(K2,2)+1
275 REM
280 NEXT K2
290 NEXT K1
293 REM
295 REM  ***  OUTPUT ROUTINE
297 REM
300 PRINT 'QUESTION','CORRECT','INCORRECT'
310 PRINT 'NUMBER','RESPONSES','RESPONSES','OMITTED'
315 PRINT
320 FOR K1=1 TO 8
330 PRINT K1,
340 FOR K2=2 TO 0 STEP -1
350 PRINT T(K1,K2),
360 NEXT K2
370 PRINT
380 NEXT K1
385 PRINT
400 REM
410 REM  ***  DATA TABLE
420 REM
430 REM     THE FIRST DATA STATEMENT CONTAINS THE CORRECT ANSWERS:
500 DATA 2,3,3,2,1,2,1,3
505 REM     THE NEXT DATA STATEMENT INDICATES THE NUMBER OF TESTS
507 REM  TO BE SCORED:
510 DATA 10
513 REM
515 REM     THE REMAINING DATA STATEMENTS CONTAIN THE ACTUAL
517 REM  RESPONSES GIVEN:
520 DATA 2,1,3,0,3,3,2,2
530 DATA 2,3,3,2,1,1,2,2
540 DATA 1,3,2,3,1,2,0,0
550 DATA 2,1,2,1,3,2,3,3
560 DATA 2,2,3,0,1,3,3,2
570 DATA 2,3,2,1,3,2,0,2
580 DATA 1,3,3,2,1,3,1,0
590 DATA 2,1,2,0,0,1,3,3
600 DATA 2,2,3,1,1,2,3,2
610 DATA 2,3,2,3,3,2,3,1
999 END

READY
```

49.

```
.R BASIC
NEW OR OLD--NEW EX49

READY
10 READ N
30 IF N<0 THEN 90
35 IF N=0 THEN 72
50 FOR K=1 TO N
60 PRINT 'X';
70 NEXT K
71 GOTO 10
72 PRINT
73 PRINT '      ';
74 GOTO 10
90 FOR K=1 TO -N
100 PRINT ' ';
110 NEXT K
140 IF N>-20 THEN 10
150 PRINT 'PRESS RETURN:';
155 INPUT A$
200 DATA -5,6,-17,7,0,-1,4,-14,4,-5,4,0
205 DATA -1,4,-13,3,-8,3,0,-1,4,-5,5,-3,3,-8,3,0
210 DATA -1,14,-3,4,-5,4,0,15,-5,7,0
215 DATA 6,-4,5,-3,15,0,-2,3,-6,2,-6,4,-6,4,0
220 DATA -3,3,-4,2,-7,8,0,-4,3,-2,2,-8,8,0
225 DATA -5,5,-9,4,-6,4,0,-6,3,-9,15,-20
999 END
SAVE RXA1:EX49.BA

READY
OLD RESEQ

READY
RUNNH
FILE?RXA1:EX49.BA
START,STEP?1000,10

READY
OLD RXA1:EX49

READY
LIST

EX49    BA    3.0     16-SEP-76

1000 READ N
1010 IF N<0 THEN 1100
1020 IF N=0 THEN 1070
1030 FOR K=1 TO N
1040 PRINT 'X';
1050 NEXT K
1060 GOTO 1000
1070 PRINT
1080 PRINT '      ';
1090 GOTO 1000
1100 FOR K=1 TO -N
1110 PRINT ' ';
1120 NEXT K
1130 IF N>-20 THEN 1000
1140 PRINT 'PRESS RETURN:';
1150 INPUT A$
1160 DATA -5,6,-17,7,0,-1,4,-14,4,-5,4,0
1170 DATA -1,4,-13,3,-8,3,0,-1,4,-5,5,-3,3,-8,3,0
1180 DATA -1,14,-3,4,-5,4,0,15,-5,7,0
1190 DATA 6,-4,5,-3,15,0,-2,3,-6,2,-6,4,-6,4,0
1200 DATA -3,3,-4,2,-7,8,0,-4,3,-2,2,-8,8,0
1210 DATA -5,5,-9,4,-6,4,0,-6,3,-9,15,-20
1220 END

READY
```

50.

```
LISTNH
1000 READ N \ IF N<0 THEN 1100 \ IF N=0 THEN 1070
1030 FOR K=1 TO N \ PRINT 'X'; \ NEXT K \ GOTO 1000
1070 PRINT \ PRINT '      '; \ GOTO 1000
1100 FOR K=1 TO -N \ PRINT ' '; \ NEXT K
1130 IF N>-20 THEN 1000 \ PRINT 'PRESS RETURN:'; \ INPUT A$
1160 DATA -5,6,-17,7,0,-1,4,-14,4,-5,4,0
1170 DATA -1,4,-13,3,-8,3,0,-1,4,-5,5,-3,3,-9,3,0
1180 DATA -1,14,-3,4,-5,4,0,15,-5,7,0
1190 DATA 6,-4,5,-3,15,0,-2,3,-6,2,-6,4,-6,4,0
1200 DATA -3,3,-4,2,-7,8,0,-4,3,-2,2,-8,8,0
1210 DATA -5,5,-9,4,-6,4,0,-6,3,-9,15,-20
1220 END

READY
RUNNH
     XXXXXX                  XXXXXXX
      XXXX               XXXX     XXXX
      XXXX              XXX        XXX
      XXXX     XXXXX    XXX         XXX
      XXXXXXXXXXXXXX    XXXX     XXXX
     XXXXXXXXXXXXXXX      XXXXXXX
     XXXXXX    XXXXX    XXXXXXXXXXXXXXX
       XXX      XX       XXXX      XXXX
        XXX    XX        XXXXXXXX
         XXX  XX         XXXXXXXX
          XXXXX          XXXX      XXXX
           XXX          XXXXXXXXXXXXXXX
READY

BYE
```

51.

```
100 PRINT "UNSORTED DATA:"
110 FOR K=1 TO 10 \ READ N(K) \ NEXT K
```

```
120 GOSUB 700
```

```
150 PRINT
160 PRINT
170 FOR K1=1 TO 9
180 FOR K2=K1+1 TO 10
190 IF N(K1)<=N(K2) THEN 230
200 LET T=N(K1)
210 LET N(K1)=N(K2)
220 LET N(K2)=T
230 NEXT K2
240 NEXT K1
250 PRINT "SORTED DATA"
```

```
260 GOSUB 700
```

```
270 STOP
500 DATA 66,75,59,93,77,85,48,92,67,78
```

```
700 FOR K=1 TO 10
710 PRINT N(K);
720 NEXT K
730 RETURN
```

```
999 END

READY
RUNNH
UNSORTED DATA:
 66   75   59   93   77   85   48   92   67   78

SORTED DATA
 48   59   66   67   75   77   78   85   92   93
READY
```

52.

```
10 PRINT "A","C","B"
20 READ A,C
30 LET B=SQR(C^2-A^2)
40 PRINT A,C,B
50 GOTO 20
90 DATA 1,2,2,3,3.6,4.7
99 END

READY
RUNNH
A              C              B
 1              2              1.73205
 2              3              2.23607
 3.6            4.7            3.02159

DA AT LINE 00020

READY
```

53.

```
10 PRINT "YOUR NUMBER";
20 INPUT N
30 PRINT "ANSWER = "; -1*SGN(N)*N^2
40 GOTO 10
99 END

READY
RUNNH
YOUR NUMBER?6
ANSWER = -36
YOUR NUMBER?-2
ANSWER =  4
YOUR NUMBER?3.14
ANSWER = -9.85959
YOUR NUMBER?^C
READY
```

54.

```
10 INPUT X
20 PRINT INT(X+0.5)
30 GOTO 10
99 END
RUNNH
?2.1
 2
?26.8
 27
?-3.3
-3
?-126.7
-127
?^C
READY
```

55. Rounding to the nearest **tenth:**

```
10 INPUT X
20 PRINT INT(10*X+0.5)/10
30 GOTO 10
99 END

READY
RUNNH
?15.21
 15.2
?-121.03
-121
?2.617
 2.6
?^C
READY
```

Rounding to the nearest **hundredth:**

```
10 INPUT X
20 PRINT INT(100*X+0)/100
30 GOTO 10
99 END

READY
RUNNH
?1.3156
 1.31
?66.449
 66.44
?-15.326
-15.33
?^C
READY
```

Rounding to the nearest **ten:**

```
20 PRINT 10*INT(X/10+0.5)
LISTNH

10 INPUT X
20 PRINT 10*INT(X/10+0.5)
30 GOTO 10
99 END

READY
RUNNH
?55
 60
?224.5
 220
?-77
-80
?-80
-80
?^C
READY
```

Rounding to the nearest **hundred:**

```
10 INPUT X
20 PRINT 100*INT(X/100+0.5)
30 GOTO 10
99 END

READY
RUNNH
?1.6
 0
?450.02
 500
?-270
-300
?^C
READY
```

56.

```
10 READ X
20 LET Y=INT(X/10)+10*(X/10-INT(X/10))
25 REM
30 PRINT X,Y
40 GOTO 10
90 DATA 10,15,23,37,40,99
99 END

READY
RUNNH
 10             1
 15             6
 23             5
 37             10
 40             4
 99             18

DA AT LINE 00010

READY
```

57.

```
10 READ X
20 LET Y=INT(X/10)+100*(X/10-INT(X/10))
30 PRINT X,Y
40 GOTO 10
90 DATA 10,37,99
99 END

READY
RUNNH
 10             1
 37             73
 99             98.9999

DA AT LINE 00010

READY
```

58.

```
10 PRINT "HYDROGEN ION CONCENTRATION";
20 INPUT C
30 PRINT "PH ="; -LOG(C)/LOG(10)
40 GOTO 10
99 END

READY
RUNNH
HYDROGEN ION CONCENTRATION?.0000001
PH = 7
HYDROGEN ION CONCENTRATION?.00003875
PH = 4.41173
HYDROGEN ION CONCENTRATION?.0000000387
PH = 8.41229
HYDROGEN ION CONCENTRATION?1.24E-10
PH = 9.90658
HYDROGEN ION CONCENTRATION?2.77E-2
PH = 1.55752
HYDROGEN ION CONCENTRATION?^C
READY
```

59.

```
10 PRINT 'PH';
20 INPUT P
30 PRINT 'HYDROGEN ION CONCENTRATION ='; EXP(-P*LOG(10))
40 GOTO 10
99 END

READY
RUNNH
PH?7
HYDROGEN ION CONCENTRATION = .100001E-006
PH?4.41173
HYDROGEN ION CONCENTRATION = 0.00003875
PH?8.41229
HYDROGEN ION CONCENTRATION = .387006E-008
PH?9.90658
HYDROGEN ION CONCENTRATION = .124002E-009
PH?1.55752
HYDROGEN ION CONCENTRATION = 0.0277001
PH?^C
READY
```

60.

```
100 PRINT
110 PRINT 'ANGLE','SINE','COSINE','TANGENT','ANGLE'
120 PRINT
130 LET P=180
140 FOR K=0 TO 4*P STEP P/4
150 LET A=3.14159*K/180
160 PRINT K,SIN(A), COS(A),
170 LET T=SIN(A)/COS(A)
180 PRINT T,180*ATN(T)/3.14159
190 NEXT K
200 END

READY
RUNNH
```

ANGLE	SINE	COSINE	TANGENT	ANGLE
0	0	0.999999	0	0
45	0.707106	0.707108	0.999998	45
90	0.999999	0.0000015	667544	90
135	0.707109	-0.707105	-1	-45.0002
180	0.00000337	-0.999999	-0.00000337	-0.00019312
225	-0.707104	-0.70711	0.999991	44.9998
270	-0.999999	-0.00000599	166886	89.9997
315	-0.707112	0.707102	-1.00001	-45.0005
360	-0.00000674	0.999999	-0.00000674	-0.00038624
405	0.707102	0.707112	0.999985	44.9996
450	0.999999	0.00001049	95363.4	89.9995
495	0.707114	-0.707098	-1.00002	-45.0007
540	0.00001198	-0.999999	-0.00001198	-0.00068665
585	-0.707098	-0.707115	0.999976	44.9993
630	-0.999999	-0.00001348	74171.6	89.9993
675	-0.707118	0.707096	-1.00003	-45.0009
720	-0.00001348	0.999999	-0.00001348	-0.00077248

READY

61.

```
10 PRINT 'DISTANCE IN METERS';
20 INPUT D
30 PRINT 'ANGLE IN DEGREES';
40 INPUT A
50 LET A=3.14159*A/180
60 PRINT 'HEIGHT = ';D*SIN(A)/COS(A);'METERS'
70 PRINT
80 GOTO 10
99 END

READY
RUNNH
DISTANCE IN METERS?50
ANGLE IN DEGREES?60
HEIGHT =  86.6022 METERS

DISTANCE IN METERS?126.3
ANGLE IN DEGREES?48.5
HEIGHT =  142.756 METERS

DISTANCE IN METERS?85.37
ANGLE IN DEGREES?1.90
HEIGHT =  2.83201 METERS

DISTANCE IN METERS?^C
READY
```

62.

(1) [0,1,2]
(2) [0,1,2,3,4,5]
(3) [1,2,3,4,5,6]
(4) [0,1,2,3,4,5,6,7,8,9]
(5) [0,.1,.2,.3,.4,.5,.6,.7,.8,.9]

63.

```
100 REM *** INITIALIZATION
110 REM
120 DIM T(12)
130 FOR K=1 TO 12 \ LET T(K)=0\NEXT K
140 PRINT 'TOTAL DOTS SHOWN','PERCENT OCCURRENCE'
150 RANDOMIZE
160 REM
170 REM  ***  SIMULATION
180 REM
190 FOR K=1 TO 1000
200 LET A=1+INT(6*RND(0))
210 LET B=1+INT(6*RND(0))
220 LET T(A+B)=T(A+B)+1
230 NEXT K
240 REM
250 REM  ***  OUTPUT
260 REM
270 FOR K=2 TO 12 \ PRINT K,, T(K)/10 \ NEXT K
280 END

READY
RUNNH
TOTAL DOTS SHOWN            PERCENT OCCURRENCE
 2                            3.2
 3                            5.6
 4                            7.7
 5                            12.6
 6                            13.5
 7                            14.2
 8                            15.6
 9                            11
 10                           7.5
 11                           6.5
 12                           2.6

READY
RUNNH
TOTAL DOTS SHOWN            PERCENT OCCURRENCE
 2                            2.8
 3                            5.2
 4                            9
 5                            11.4
 6                            13.1
 7                            16.2
 8                            14.2
 9                            10.8
 10                           9
 11                           5.7
 12                           2.6

READY
```

64.

```
100 REM        ----------------
110 REM  ***  CIRCLE FUNCTIONS
120 REM        ----------------
130 REM
140 REM  ***  FUNCTION DEFINITION
150 REM
160 DEF FNP(X)=3.14159*R
170 REM
180 REM  ***  DATA INPUT
190 REM
200 PRINT 'RADIUS';
210 INPUT R
220 REM
230 REM  ***  OUTPUT
240 REM
250 PRINT 'RADIUS','CIRCUMFERENCE','CIRCLE AREA','SURFACE AREA','VOLUME'
260 PRINT R, 2*FNP(R), R*FNP(R), 4*R*FNP(R), 4*R^2*FNP(R)/3
270 PRINT
280 GOTO 200
290 END
```

continued on next page

```
READY
RUNNH
RADIUS?42
RADIUS        CIRCUMFERENCE CIRCLE AREA   SURFACE AREA   VOLUME
 42            263.893       5541.76       22167          310338

RADIUS?2.315
RADIUS        CIRCUMFERENCE CIRCLE AREA   SURFACE AREA   VOLUME
 2.315         14.5455       16.8365       67.3459        51.9686

RADIUS?3E10
RADIUS        CIRCUMFERENCE CIRCLE AREA   SURFACE AREA   VOLUME
 .300000E+011  .188495E+012  .282742E+022  .113097E+023   .113097E+033

RADIUS?.0000218
RADIUS        CIRCUMFERENCE CIRCLE AREA   SURFACE AREA   VOLUME
 0.0000218     0.00013697    .149301E-008  .597203E-008   .433967E-013

RADIUS?^C
READY
```

65.

```
100 REM       -----------------------
110 REM  ***  23 MATCHES SCORE KEEPER
120 REM       -----------------------
130 REM
140 REM  ***  DIMENSION STRINGS
150 REM
160 DIM N$(10,20)
170 REM
180 REM  ***  GET NAMES
190 REM
200 PRINT 'HOW MANY PEOPLE WILL PLAY';
210 INPUT N
220 PRINT
230 FOR K=1 TO N
240 PRINT 'WHAT IS PLAYER #'; K; "'S NAME';
250 INPUT N$(K)
260 NEXT K
270 REM
280 REM  ***  PLAY THE GAME
290 REM
300 PRINT
310 LET M=23
320 FOR K=1 TO N
330 PRINT
340 PRINT N$(K); "'S TURN:'
350 PRINT 'THERE ARE NOW'; M; 'MATCHES.'
360 PRINT 'HOW MANY DO YOU WISH TO TAKE';
370 INPUT H
380 IF H<=0 THEN 420
390 IF H>=4 THEN 420
400 IF H>M THEN 420
410 IF H=INT(H) THEN 440
420 PRINT 'YOU CHEATED!  I'LL GIVE YOU ANOTHER CHANCE:...'
430 GOTO 330
440 LET M=M-H
450 IF M=0 THEN 520
460 IF M=1 THEN 560
470 NEXT K
480 GOTO 320
490 REM
500 REM  ***  SOMEBODY LOST
510 REM
520 PRINT \ PRINT
530 PRINT N$(K); ' LOST THIS TIME!'
540 LET K=K-1
550 GOTO 630
560 PRINT \ PRINT
570 IF K<>N THEN 590
580 LET K=0
590 PRINT N$(K+1); ' MUST TAKE THE LAST MATCH!'
600 REM
610 REM  ***  PRINT SCORES
620 REM
630 LET S(K+1)=S(K+1)+1
640 PRINT \ PRINT
650 PRINT 'THE SCORE IS NOW:'
660 PRINT 'NAME','NUMBER OF GAMES LOST'
670 FOR K=1 TO N
680 PRINT N$(K), S(K)
690 NEXT K
700 REM  ***  RERUN QUERY
710 REM
720 PRINT \ PRINT
730 PRINT 'DO YOU WANT TO PLAY AGAIN (''YES'' OR ''NO'')';
740 INPUT A$
750 IF A$='YES' THEN 300
760 IF A$='NO' THEN 790
770 PRINT 'PLEASE ANSWER ONLY ''YES'' OR ''NO''.'
780 GOTO 730
790 END
READY
RUNNH
HOW MANY PEOPLE WILL PLAY?2

WHAT IS PLAYER # 1 'S NAME?HILLEL
WHAT IS PLAYER # 2 'S NAME?JESSE
```

continued on next column

```
HILLEL'S TURN:
THERE ARE NOW 23 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

JESSE'S TURN:
THERE ARE NOW 21 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

HILLEL'S TURN:
THERE ARE NOW 19 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

JESSE'S TURN:
THERE ARE NOW 17 MATCHES.
HOW MANY DO YOU WISH TO TAKE?1

HILLEL'S TURN:
THERE ARE NOW 16 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

JESSE'S TURN:
THERE ARE NOW 14 MATCHES.
HOW MANY DO YOU WISH TO TAKE?1

HILLEL'S TURN:
THERE ARE NOW 13 MATCHES.
HOW MANY DO YOU WISH TO TAKE?3

JESSE'S TURN:
THERE ARE NOW 10 MATCHES.
HOW MANY DO YOU WISH TO TAKE?1

HILLEL'S TURN:
THERE ARE NOW 9 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

JESSE'S TURN:
THERE ARE NOW 7 MATCHES.
HOW MANY DO YOU WISH TO TAKE?2

HILLEL'S TURN:
THERE ARE NOW 5 MATCHES.
HOW MANY DO YOU WISH TO TAKE?1

JESSE'S TURN:
THERE ARE NOW 4 MATCHES.
HOW MANY DO YOU WISH TO TAKE?3


HILLEL MUST TAKE THE LAST MATCH!


THE SCORE IS NOW:
NAME           NUMBER OF GAMES LOST
HILLEL          1
JESSE           0


DO YOU WANT TO PLAY AGAIN ('YES' OR 'NO')?NO

READY
```

66.

```
10 DIM S$(72)
20 LET S$=""
30 INPUT T,A$
40 FOR K=1 TO T
50 LET S$=S$ & A$
60 NEXT K
70 PRINT S$
80 GOTO 20
99 END
READY
RUNNH
?4 HI
HIHIHIHI
?2 LOW
LOWLOW
?3 DIGI
DIGIDIGIDIGI
?^C
READY
```

67.

```
100 REM   ***   23 MATCHES
110 LET M=23
120 PRINT
130 PRINT "WE START WITH 23 MATCHES.  WHEN IT IS YOUR TURN, YOU MAY "
140 PRINT "TAKE 1, 2, OR 3"
150 PRINT "MATCHES.  THE ONE WHO MUST TAKE THE LAST MATCH LOSES."
160 PRINT
170 REM   ***   THE HUMAN MOVES
180 PRINT
190 IF M>1 THEN 220
200 PRINT "THERE IS NOW ONLY 1 MATCH LEFT."
210 GOTO 230
220 PRINT "THERE ARE NOW"; M; "MATCHES."
230 PRINT
240 PRINT "HOW MANY MATCHES DO YOU WISH TO TAKE";
242 INPUT H$
244 IF H$<>"UNCLE" THEN 254
245 PRINT
246 PRINT "O.K., LET'S START AGAIN..."
248 PRINT
250 LET M=23
252 GOTO 180
254 LET H=VAL(H$)
260 IF H>M THEN 320
270 IF H=INT(H) THEN 290
280 GOTO 320
290 IF H>0 THEN 310
300 GOTO 320
310 IF H<4 THEN 340
320 PRINT "YOU CHEATED!  I'LL GIVE YOU ANOTHER CHANCE:"
330 GOTO 230
340 LET M=M-H
350 IF M=0 THEN 530
360 REM
370 REM   ***   THE COMPUTER MOVES
380 REM
390 LET R=M-4*INT(M/4)
400 IF M=1 THEN 580
410 IF R<>1 THEN 440
420 LET C=INT(3*RND(0))+1
430 GOTO 450
440 LET C=(M+3)-4*INT((M+3)/4)
450 LET M=M-C
460 IF M=0 THEN 580
470 PRINT
480 PRINT "I TOOK"; C; "MATCHES."
490 RANDOMIZE
500 GOTO 180
510 REM   ***   SOMEBODY WON
520 REM
530 PRINT
540 PRINT
550 PRINT "I WON!  BETTER LUCK NEXT TIME..."
560 LET M=23
570 GOTO 160
580 PRINT
590 PRINT
600 PRINT "CONGRATULATIONS!  YOU MADE ME TAKE THE LAST MATCH."
610 PRINT
620 PRINT "YOU WON THIS TIME, BUT LET'S PLAY AGAIN..."
630 GOTO 560
640 END

READY
RUNNH

WE START WITH 23 MATCHES.  WHEN IT IS YOUR TURN, YOU MAY TAKE 1, 2, OR 3
MATCHES.  THE ONE WHO MUST TAKE THE LAST MATCH LOSES.


THERE ARE NOW 23 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?2

I TOOK 2 MATCHES.

THERE ARE NOW 19 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?3

I TOOK 3 MATCHES.

THERE ARE NOW 13 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?UNCLE

O.K., LET'S START AGAIN...


THERE ARE NOW 23 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?2

I TOOK 3 MATCHES.

THERE ARE NOW 18 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?3

I TOOK 2 MATCHES.

THERE ARE NOW 13 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?2

I TOOK 2 MATCHES.

THERE ARE NOW 9 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?1

I TOOK 3 MATCHES.

THERE ARE NOW 5 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?2

I TOOK 2 MATCHES.

THERE IS NOW ONLY 1 MATCH LEFT.

HOW MANY MATCHES DO YOU WISH TO TAKE?1


I WON!  BETTER LUCK NEXT TIME...


THERE ARE NOW 23 MATCHES.

HOW MANY MATCHES DO YOU WISH TO TAKE?^C
READY
```

68.

N	LEN(STR$(N))	6-LEN(STR$(N))
47	2	4
126	3	3
8	1	5
2873	4	2
61045	5	1

```
10 READ N
20 PRINT N,LEN(STR$(N)),6-LEN(STR$(N))
30 GOTO 10
40 DATA 47,126,8,2873,61045
99 END

READY
RUNNH
 47            2             4
 126           3             3
 8             1             5
 2873          4             2
 61045         5             1

DA AT LINE 00010

READY
```

69

```
10 PRINT "N","SQUARE ROOT","FOURTH ROOT"
20 LET N=.1
30 FOR K=1 TO 7

40 LET S=N
45 GOSUB 100
50 LET S=SQR(N)
55 GOSUB 100
60 LET S=SQR(SQR(N))
63 GOSUB 100
65 PRINT

70 LET N=INT(N*10+.5)
80 NEXT K
90 STOP
100 FOR KO=1 TO 6-LEN(STR$(INT(S)))
110 PRINT " ";
120 NEXT KO
130 PRINT STR$(S),
140 RETURN

999 END
```

```
READY
RUNNH
N              SQUARE ROOT   FOURTH ROOT
     0.1          0.316228      0.562341
     1            1             1
    10            3.16228       1.77828
   100           10             3.16228
  1000           31.6228        5.62341
 10000          100            10
100000          316.228        17.7828

READY
```

71.

```
5 DEF FNF(X)=7-LEN(STR$(INT(X)))

10 PRINT "N","SQUARE ROOT","FOURTH ROOT"
20 LET N=.1
30 FOR K=1 TO 7

40 PRINT TAB(FNF(N)); STR$(N);
50 PRINT TAB(14+FNF(SQR(N))); STR$(SQR(N));
60 PRINT TAB(28+FNF(SQR(SQR(N)))); STR$(SQR(SQR(N)))

70 LET N=INT(N*10+.5)
80 NEXT K
99 END

READY
RUNNH
N              SQUARE ROOT   FOURTH ROOT
     0.1          0.316228      0.562341
     1            1             1
    10            3.16228       1.77828
   100           10             3.16228
  1000           31.6228        5.62341
 10000          100            10
100000          316.228        17.7828

READY
```

72.

```
10 GOSUB 80
20 FOR K=1 TO 8

30 LET T=36+((4-K)/2)^3

40 PRINT TAB(T);"*"
50 NEXT K
60 GOSUB 80
70  STOP
80 FOR K1=1 TO 7 \ FOR K2=1  TO 9 \ PRINT STR$(K2); \ NEXT K2
90 PRINT "-"; \ NEXT K1 \ PRINT "12" \ RETURN
99 END

READY
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
                                      *
                                    *
                                   *
                                   *
                                  *
                                  *
                               *
                           *
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
30 LET T=72*RND(0)
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
                         *
                      *
                                            *
                        *
                                               *
                                      *
                                                            *
*
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
30 LET T=2*T+1
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
*
  *
      *
              *
                              *
                                                              *


*
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
30 LET T=36-(4-K)^2
RUNNH
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12
                          *
                               *
                                  *
                                   *
                                  *
                               *
                          *
                   *
123456789-123456789-123456789-123456789-123456789-123456789-123456789-12

READY
```

77.

```
80 DIM A$(100,20), T$(20), X$(20), Y$(20)
90 READ N
100 FOR K=1 TO N \ READ A$(K) \ NEXT K
120 FOR K1=1 TO N-1
130 FOR K2=K1+1 TO N

140 IF LEN(A$(K1))=LEN(A$(K2)) THEN 240
150 LET X$=A$(K1)
155 LET Y$=A$(K2)
160 IF LEN(X$)>LEN(Y$) THEN 210
170 FOR K=1 TO LEN(Y$)-LEN(X$)
180 LET X$=X$ & "@"
190 NEXT K
200 GOTO 230
210 FOR K=1 TO LEN(X$)-LEN(Y$)
220 LET Y$=Y$ & "@"
225 NEXT K
230 IF X$<Y$ THEN 280
235 GOTO 250
240 IF A$(K1)<A$(K2) THEN 280
250 LET T$=A$(K1)
260 LET A$(K1)=A$(K2)
270 LET A$(K2)=T$
280 NEXT K2
290 NEXT K1
300 PRINT \ PRINT "SORTED DATA:" \ PRINT
310 FOR K=1 TO N/3+1
320 PRINT A$(K),, A$(K+1+INT(N/3)),, A$(K+2+2*INT(N/3))
330 NEXT K

500 DATA 50
510 DATA "MISSOURI", "CONNECTICUT", "NEW MEXICO", "WISCONSIN"
520 DATA "ALABAMA", "MARYLAND", "TENNESSEE", "NEBRASKA", "UTAH"
530 DATA "RHODE ISLAND", "SOUTH DAKOTA", "WEST VIRGINIA"
540 DATA "MINNESOTA", "CALIFORNIA", "FLORIDA", "OHIO"
550 DATA "KENTUCKY", "INDIANA", "MASSACHUSETTS", "OREGON"
560 DATA "NEW JERSEY", "VIRGINIA", "MAINE", "DELEWARE"
570 DATA "IDAHO", "ILLINOIS", "NORTH CAROLINA", "ALASKA"
580 DATA "NEW YORK", "TEXAS", "GEORGIA", "KANSAS", "WYOMING"
590 DATA "WASHINGTON", "MICHIGAN", "MISSISSIPPI", "IOWA"
600 DATA "VERMONT", "MONTANA", "PENNSYLVANIA", "HAWAII"
610 DATA "LOUISIANA", "COLORADO", "OKLAHOMA", "SOUTH CAROLINA"
620 DATA "ARIZONA", "ARKANSAS", "NORTH DAKOTA", "NEVADA"
630 DATA "NEW HAMPSHIRE"
999 END

READY
RUNNH

SORTED DATA:

ALABAMA                     LOUISIANA                   OHIO
ALASKA                      MAINE                       OKLAHOMA
ARIZONA                     MARYLAND                    OREGON
ARKANSAS                    MASSACHUSETTS               PENNSYLVANIA
CALIFORNIA                  MICHIGAN                    RHODE ISLAND
COLORADO                    MINNESOTA                   SOUTH CAROLINA
CONNECTICUT                 MISSISSIPPI                 SOUTH DAKOTA
DELEWARE                    MISSOURI                    TENNESSEE
FLORIDA                     MONTANA                     TEXAS
GEORGIA                     NEBRASKA                    UTAH
HAWAII                      NEVADA                      VERMONT
IDAHO                       NEW HAMPSHIRE               VIRGINIA
ILLINOIS                    NEW JERSEY                  WASHINGTON
INDIANA                     NEW MEXICO                  WEST VIRGINIA
IOWA                        NEW YORK                    WISCONSIN
KANSAS                      NORTH CAROLINA              WYOMING
KENTUCKY                    NORTH DAKOTA

READY
```

81.

```
100 DIM A$(72)
120 PRINT \ PRINT "THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC."
140 RANDOMIZE
160 LET A=INT(10*RND(0))
180 LET B=INT(10*RND(0))
200 PRINT \ PRINT "HOW MUCH IS"; A; "+"; B;
220 INPUT A$
240 LET C$=STR$(A+B)
260 IF POS(A$,"QUIT",1)>0 THEN 560
280 IF POS(A$,"HELP",1)>0 THEN 520
300 IF POS(A$,C$,1)>0 THEN 380
320 PRINT "  INCORRECT.  PLEASE TRY AGAIN..."
340 GOTO 200
380 LET P=POS(A$,C$,1)
400 IF SEG$(A$,P-1,P-1)<>"-" THEN 440
420 GOTO 320
440 IF POS(A$,"NOT",1)=0 THEN 480
460 IF POS(A$,"NOT",1)<POS(A$,C$,1) THEN 320

480 IF POS(A$,"N'T",1)=0 THEN 490
485 IF POS(A$,"N'T",1)<POS(A$,C$,1) THEN 320

490 PRINT "  CORRECT!"
500 GOTO 160
520 PRINT " "; A; "+"; B; "="; A+B; ".  HERE'S ANOTHER..."
540 GOTO 160
560 END

READY
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 4 + 5 ?IT ISN'T 9
  INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 4 + 5 ?I GUESS IT MUST BE 9
  CORRECT!

HOW MUCH IS 2 + 5 ?I'M QUITTING NOW!!!

READY
```

82.

```
100 DIM A$(72)
120 PRINT \ PRINT "THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC."
140 RANDOMIZE
160 LET A=INT(10*RND(0))
180 LET B=INT(10*RND(0))
200 PRINT \ PRINT "HOW MUCH IS"; A; "+"; B;
220 INPUT A$
240 LET C$=STR$(A+B)
260 IF POS(A$,"QUIT",1)>0 THEN 560
280 IF POS(A$,"HELP",1)>0 THEN 520
300 IF POS(A$,C$,1)>0 THEN 380
320 PRINT "  INCORRECT.  PLEASE TRY AGAIN..."
340 GOTO 200
380 LET P=POS(A$,C$,1)
400 IF SEG$(A$,P-1,P-1)<>"-" THEN 440
420 GOTO 320
440 IF POS(A$,"NOT",1)=0 THEN 480
460 IF POS(A$,"NOT",1)<POS(A$,C$,1) THEN 320
480 IF POS(A$,"N'T",1)=0 THEN 490
485 IF POS(A$,"N'T",1)<POS(A$,C$,1) THEN 320

490 IF ASC(SEG$(A$,P-1,P-1))<48 THEN 493
491 IF ASC(SEG$(A$,P-1,P-1))>57 THEN 493
492 GOTO 320
493 LET P1=P+LEN(C$)
494 IF ASC(SEG$(A$,P1,P1))<48 THEN 496
495 IF ASC(SEG$(A$,P1,P1))<57 THEN 320
496 PRINT "  CORRECT!"

500 GOTO 160
520 PRINT " "; A; "+"; B; "="; A+B; ".  HERE'S ANOTHER..."
540 GOTO 160
560 END

READY
RUNNH

THIS PROGRAM HELPS YOU PRACTICE ARITHMETIC.

HOW MUCH IS 3 + 4 ?THAT'S 7
  CORRECT!

HOW MUCH IS 4 + 4 ?123456789101112131415161718
  INCORRECT.  PLEASE TRY AGAIN...

HOW MUCH IS 4 + 4 ?WELL, I GUESS IT MUST BE 8
  CORRECT!

HOW MUCH IS 3 + 0 ?THAT'S ENOUGH, I QUIT!

READY
```

85.

```
10 FILEV #1: "RXA1:JABBER.LC"
20 PRINT #1: "'TWAS BRILLIG, AND THE SLITHY TOVES"
30 PRINT #1: "  DID GYRE AND GIMBEL IN THE WABE"
40 PRINT #1:"ALL MIMSEY WERE THE BOROGROVES"
50 PRINT #1:"  AND THE MOME RATHS OUTGRABE"
60 CLOSE #1
70 PRINT "JABBERWOCKY FILE CREATED"
99 END

READY
RUNNH
JABBERWOCKY FILE CREATED

READY
BYE

.TYPE RXA1:JABBER.LC
'TWAS BRILLIG, AND THE SLITHY TOVES
  DID GYRE AND GIMBEL IN THE WABE
ALL MIMSEY WERE THE BOROGROVES
  AND THE MOME RATHS OUTGRABE

.R BASIC
NEW OR OLD--NEW RXA1:X85A

READY

10 FILE #1: "RXA1:JABBER.LC"
20 DIM L$(72)
30 FOR K=1 TO 4
40 INPUT #1: L$
50 PRINT L$
60 NEXT K
99 END

READY
RUNNH
'TWAS BRILLIG AND THE SLITHY TOVES
   DID GYRE AND GIMBLE IN THE WABE;
ALL MIMSY WERE THE BOROGOVES,
   AND THE MOME RATHS OUTGRABE.

READY
```

86.

```
LIST

10 FILEV #1: "RXA1:OUTPUT.JH"
20 PRINT #1: "THIS DATA IS STORED IN A DISK FILE."
25 PRINT #1: "THIS IS THE SECOND LINE OF THE DATA."
30 CLOSE #1
40 DIM A$(72)
50 FILE #1: "RXA1:OUTPUT.JH"
60 INPUT #1: A$
65 IF END #1 THEN 90
70 PRINT A$
80 GOTO 60

90 RESTORE #1
95 GOTO 60

99 END

READY
RUNNH
THIS DATA IS STORED IN A DISK FILE.
THIS IS THE SECOND LINE OF THE DATA.
THIS DATA IS STORED IN A DISK FILE.
THIS IS THE SECOND LINE OF THE DATA.
THIS DATA IS STORED IN A DISK FILE.
THIS IS THE SECOND LINE OF THE DATA.
THIS DA^C
READY
```

88. All three arrangements produce the same results:

```
100 LET K=10
110 FILEVN #1: "RXA1:TEST"
120 PRINT #1: K, K+1, K+2
130 CLOSE #1
140 FILEN #1: "RXA1:TEST"
150 INPUT #1: A,B,C
160 PRINT A;B;C
999 END

READY
RUNNH
 10   11   12

READY
120 PRINT #1:K, K+1, K+2
RUNNH
 10   11   12

READY
120 PRINT #1: K
123 PRINT #1: K+1
125 PRINT #1: K+2
RUNNH
 10   11   12

READY
```

89.

```
100 REM  ***  TALLY2
110 REM
120 REM     THIS PROGRAM WRITES THE DATA FILE FOR TALLY1.
130 REM
140 FILEVN #1: 'RXA1:QUESTI.ON'
150 READ N
160 PRINT #1: N
170 FOR K1=1 TO N
180 FOR K2=1 TO 8
190 READ R
200 PRINT #1: R
210 NEXT K2
220 NEXT K1
230 CLOSE #1
240 PRINT 'QUESTION DATA FILE WRITTEN.'
510 REM
520 REM  ***  DATA TABLE
530 REM
540 DATA 5
550 REM     THE FIRST DATA ITEM INDICATES THE NUMBER OF
560 REM  SURVEYS TO BE TALLIED.  THE ACTUAL RESPONSES
570 REM  GIVEN FOLLOW BELOW:
580 DATA 2,1,4,4,1,3,1,4
590 DATA 1,1,3,4,2,4,1,3
600 DATA 2,1,4,4,1,3,1,3
610 DATA 2,2,4,4,1,3,2,1
620 DATA 4,3,4,1,3,3,2,2
630 END

READY
RUNNH
QUESTION DATA FILE WRITTEN.

READY
```

```
100 REM  ***  TALLY
110 REM
120 REM     THIS PROGRAM TALLIES THE NUMBER OF PEOPLE
130 REM  CHOOSING EACH OF 4 RESPONSES FOR EACH OF 8
140 REM  QUESTIONS.
150 REM
160 DIM T(8,4)
170 REM  ARRAY 'T' STORES THE TALLY COUNTS
180 REM
190 REM  ***  INITIALIZING ROUTINE
200 REM
210 FOR K1=1 TO 8
220 FOR K2=1 TO 4
230 LET T(K1,K2)=0
240 NEXT K2
250 NEXT K1
255 REM
260 FILEN #1: 'RXA1:QUESTI.ON'
270 INPUT #1: N
275 REM
280 REM  'N' IS THE NUMBER OF SURVEYS TO BE TALLIED.
290 REM
300 REM  ***  TALLYING ROUTINE
310 REM
320 FOR K1=1 TO N
330 FOR K2=1 TO 8
335 REM
340 INPUT #1: R
345 REM
350 REM  'R' IS A RESPONSE TO QUESTION NUMBER 'K2'.
360 LET T(K2,R)=T(K2,R)+1
370 NEXT K2
380 NEXT K1
390 REM
400 REM  ***  OUTPUT ROUTINE
410 REM
420 PRINT "QUESTION",,'RESPONSES"
430 PRINT ,'1','2','3','4'
440 REM
450 FOR K1=1 TO 8
460 PRINT K1,
470 FOR K2=1 TO 4
480 PRINT T(K1,K2),
490 NEXT K2
495 PRINT
500 NEXT K1
630 END

READY
RUNNH
QUESTION                    RESPONSES
              1              2              3              4
 1            1              3              0              1
 2            3              1              1              0
 3            0              0              1              4
 4            1              0              0              4
 5            3              1              1              0
 6            0              0              4              1
 7            3              2              0              0
 8            1              1              2              1

READY
```

The CLASSIC Primer:
A Self-Teaching Guide
DEC-08-ECPGA-B-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________________

Organization ______________________

Street ______________________

City ______________ State ______________ Zip Code or Country ______________

If you require a written reply, please check here. ☐

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617) 897-5111

SALES AND SERVICE OFFICES

DOMESTIC — ARIZONA, Phoenix and Tucson • CALIFORNIA, Los Angeles, Monrovia, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, Metairie (New Orleans) • MASSACHUSETTS, Marlborough and Waltham • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Huntington Station, Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland, Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Portland • PENNSYLVANIA, Philadelphia (Bluebell) and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield)

INTERNATIONAL — ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Puerto Alegre, Rio de Janeiro and São Paulo • CANADA, Calgary, Halifax, Montreal Ottawa, Toronto and Vancouver • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Grenoble and Paris • GERMANY, Berlin, Cologne, Hannover, Frankfurt, Munich and Stuttgart • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • ISRAEL, Tel Aviv ITALY, Milan and Turin • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, The Hague • NEW ZEALAND, Auckland • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SPAIN, Barcelona and Madrid • SWEDEN, Stockholm • SWITZERLAND, Geneva and Zurich • TAIWAN, Taipei • UNITED KINGDOM, Birmingham, Bristol, Dublin, Edinburgh, Leeds, London, Manchester and Reading • VENEZUELA, Caracas